PTR-1151-87-2
Contract No. DAAB07-84-C-K527
March, 1987

# DESIGN & DEVELOPMENT OF AN INTELLIGENT AID FOR TACTICAL PLAN GENERATION & EVALUATION: THE INTACVAL PROTOTYPE (FINAL REPORT)

PERCEPTRONICS, INC.

INTERNATIONAL INFORMATION SYSTEMS, INC.

INTEGRATED SYSTEMS RESEARCH CORPORATION

Prepared for:

Computer Research Division
Center for Tactical Computer Systems
U.S. Army Communications-Electronics Command
Fort Monmouth, NJ 07703

VOLUME II

APPENDIX IV, APPENDIX V

PERCEPTRONICS

21111 ERWIN STREET • WOODLAND HILLS • CALIFORNIA 91367-3713 • PHONE 818-884-7572

SECURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b. RESTRICTIVE MARKINGS<br>None |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION / AVAILABILITY OF REPORT<br>Unlimited |
| 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>PTR-1151-87-2 | 5. MONITORING ORGANIZATION REPORT NUMBER(S)<br>CDRL A002, A004, A005, A006 |
| 6a. NAME OF PERFORMING ORGANIZATION<br>PERCEPTRONICS, INC.<br>6b. OFFICE SYMBOL (If applicable) | 7a. NAME OF MONITORING ORGANIZATION<br>Computer Research Division<br>Center for Tactical Computer Systems |
| 6c. ADDRESS (City, State, and ZIP Code)<br>21111 Erwin Street<br>Woodland Hills, CA 91367 | 7b. ADDRESS (City, State, and ZIP Code)<br>COMMANDER<br>U.S.Army CECOM<br>Fort Monmouth, NJ 07703 |
| 8a. NAME OF FUNDING / SPONSORING ORGANIZATION<br>CECOM<br>8b. OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>DAAB07-84-C-K527 |
| 8c. ADDRESS (City, State, and ZIP Code) | 10. SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO. |

11. TITLE (Include Security Classification)
Design and Development of an Intelligent Aid for Tactical Plan Generation & Evaluation: The INTACVAL Prototype (Final Report) Vol. II

12. PERSONAL AUTHOR(S)
Mayer Alan Brenner, Stephen J. Andriole, Gerald W. Hopple, Yekutiel Novick

| | | | |
|---|---|---|---|
| 13a. TYPE OF REPORT<br>Technical | 13b. TIME COVERED<br>FROM ____ TO ____ | 14. DATE OF REPORT (Year, Month, Day)<br>March 1987 | 15. PAGE COUNT |

16. SUPPLEMENTARY NOTATION

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Decision Aids; Intelligent Tactical Planning; Tactical Planning Aids |
| | | | |
| | | | |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document is the Final Report for the referenced contract, "AI Techniques for Plan Revision." It presents the INTACVAL aiding concept and the details of implementation of the INTACVAL system prototype. The report integrates the reporting requirements for the User's Manual (A004), Knowledge Base Manual (A005), and System Manual (A006), as well as the Final Report (A002). [illegible] artificial intelligence [illegible]

| | |
|---|---|
| 20. DISTRIBUTION / AVAILABILITY OF ABSTRACT<br>☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION<br>Unclassified |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>Mr. Harlan Black | 22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL |

PTR-1151-87-2
Contract No. DAAB07-84-C-K527
March, 1987

# DESIGN & DEVELOPMENT OF AN INTELLIGENT AID FOR TACTICAL PLAN GENERATION & EVALUATION: THE INTACVAL PROTOTYPE (FINAL REPORT)

PERCEPTRONICS, INC.

INTERNATIONAL INFORMATION SYSTEMS, INC.

INTEGRATED SYSTEMS RESEARCH CORPORATION

**Prepared for:**

Computer Research Division
Center for Tactical Computer Systems
U.S. Army Communications-Electronics Command
Fort Monmouth, NJ 07703

VOLUME II

APPENDIX IV, APPENDIX V

PERCEPTRONICS

21111 ERWIN STREET • WOODLAND HILLS • CALIFORNIA 91367-3713 • PHONE 818-884-7572

## APPENDIX IV:
## USER'S GUIDE

# APPENDIX IV:
# INTACVAL USERS' GUIDE

## 1.Ø GENERAL PRINCIPLES

A. To select an option, move the cursor to the option on the screen using the mouse, then click on the left button of the mouse.

B. To close a window, double-click on the small box in the upper left-hand corner of the window.

C. To view the continuation of the text which appears in a particular window, use the scroll bar at the right side of the window (to scroll up and down) or the scroll bar just below the window (to scroll left and right). Clicking on the arrows in a scroll bar will scroll the text in the direction of the arrow.

## 2.Ø UTILITIES

### 2.1 Desk

To select an option from the DESK menu, the user must first click on the mouse at DESK. While holding down the mouse button the user brings the cursor to the option he wishes to select (Clock, Notepad, or Calculator) and then releases the button. When finished with an option, the user should double-click on the small box in the upper left corner of the window; this will close the window.

2.1.1 Clock: Shows the present time.

2.1.2 Notepad: Allows the entry and review of notes (sections of text). These can be stored in any text file.

2.1.3 Calculator: Supplies basic calculator functions.

### 2.2 File

This option enables the user to open a new session file, reopen an old file, or delete an old file. It also allows users to save changed knowledge bases from one session to another.

To choose an option from the FILE menu, first click on the FILE prompt. While holding down the mouse button, bring the cursor to the entry you wish to choose (New Session, Previous Session, or Exit), and then release the button.

2.2.1 New Session: This option allows the user to open a new session. When a new session is opened the default knowledge base Values are used. The new session is given a record number one higher then the largest previous session number. If ten sessions already exist, the user will be prompted to first delete an old session.

2.2.2 Previous Session: This option allows the user to display a list of previous sessions on the screen. The user may then enter any of these previous sessions and change Values. The new Values will automatically saved when EXIT is selected.

When PREVIOUS SESSION is selected and the list of previous sessions appears, the user can click on a session name. If the user double-clicks on this session name, that session will automatically be opened. If the user single-clicks, OPEN and DELETE are presented for choice. Choosing OPEN opens the desired session and updates the knowledge base Values to those contained in that session. If DELETE is selected, a confirmation prompt appears. If YES is selected from this confirmation, the session is deleted. If NO is selected, the user returns to the session list.

2.2.3 Exit: This is the only way the user can exit INTACVAL. Upon selection of EXIT, the system performs an automatic save operation for the current session. The current Values for the knowledge bases are automatically saved under the current session name.

The session saved will contain a two-digit suffix with the session number that was automatically assigned. The user can automatically save only ten sessions. Before saving the eleventh session an old session must be deleted. The PREVIOUS SESSION list will appear and the user must click on the session he wishes to delete. The old session will be removed and the new session will be automatically saved under the old session name.

## 3.Ø PROCESS MODEL

### 3.1 Selectable Process Model

3.1.1 Start: Clicking on the START option will create a display of the major steps the user should follow in the operation of the aid (process model). The user can click on any one of these steps at any time to move immediately to that function.

3.1.2 Continue: By repeatedly clicking on the CONTINUE option the user will see all the information in the system displayed in an order that assists in assimilating and processing the information. This option may prove especially helpful to the novice who is less familiar with the process of generating a Course of Action (COA).

3.1.3 Finish: Selection of this option signals that the user is ready to conclude the current session. This option automatically performs the same operations as FILE followed by EXIT to save the session, as described above.

### 3.2 Components and Sub-Components

The user can select one of the process model components by clicking on it with the mouse. The user will then view all the sub-components of that component on the screen, although only one of these sub-components will be visible at any one time. To view a different sub-component, the user should click on the top bar of the desired sub-component.

If the user selects one of the COA (Courses of Action) components, the Red or Blue COA that is currently active will appear on the screen. The opposing COA and the active Knowledge Base may be then selected by clicking on the new component buttons that appear on the screen. Notice that the elements of the Knowledge Bases are color-coded, red indicating Soviet/Warsaw Pact and blue indicating U.S./NATO selections.

3.2.1 Mission: Assumptions, Intent, and Objectives are available.

3.2.2 Terrain: General and Key Terrain are available.

3.2.3 Blue Capabilities: Blue Composition/Location/Disposition, Reinforcements, and Condition are available.

3.2.4 Red Capabilities: Red Composition/Location/Disposition, Reinforcements, and Condition are available.

3.2.5 Red COA: Red COA, Blue COA, and Red Knowledge Base are available.

3.2.6 Blue COA: Blue COA, Red COA, and Blue Knowledge Base are available.

## 4.Ø MAP DISPLAY

The high-resolution video map display shows the terrain of East and West Germany where the scenario takes place, with colored icons illustrating the locations of the opposing forces. This symbology is presented in blue and red. The symbology that is graphically equivalent to the alphanumeric information shown on the PC screen is shown at all times. When the alphanumeric data is removed from the PC screen, its graphic equivalents are removed from the map screen.

### 4.1 Map Option

The user ccn select MAP from the PC menu to display and remove map symbology as well as to move around the map area, viewing the geography at higher and lower magnifications. To move from the PC screen to the video map screen the user clicks the mouse on the MAP option. The cursor will automatically appear on the video map screen. The user can continue using the mouse to move the video map cursor (represented by a red cross) around the map.

To pan the area of map view north, south, east, west, or diagonally, the user moves the cursor to the light blue border around the map. The cursor will change into an arrow pointing in the appropriate direction. By clicking with the mouse the map will scroll in the desired direction.

To "zoom in" at a desired point, the user should first position the cursor at the location he wishes to view in increased detail and then click the mouse, and then select one of the IN boxes displayed at the map border. The same procedure is followed to zoom out, except the OUT box is selected.

4.1.1 Map Menu: The menu at the left side of the map screen contains these options:

- Red Units: displays the red unit icons.
- Blue Units: displays the blue unit icons.
- Draw Region of Interest: limits region in which icons will be displayed.
- Clear Screen: remove all icons from screen.
- Draw Line: calculates and displays the distance between the two points chosen.
- Data: displays data corresponding to the icon selected. The user must first click on the icon of interest.
- PC: returns the cursor to the PC screen.

## 5.Ø KNOWLEDGE BASE

The user can choose a new Value for any non-shaded cell of the knowledge base by clicking on that cell with the mouse. This will cause the cell containing that Value to be filled with red (if the Red Knowledge Base is active) or blue (if the Blue Knowledge Base is active). To view areas of the knowledge base not currently visible on the screen, the user may click on the horizontal or vertical scroll bars. Closing other windows that are on the screen will allow a larger area of the knowledge base to be viewed.

INTACVAL begins a new session with a series of default Values set that result in Red COA1 and Blue COA1 being displayed. As described in Appendix II, one set of new Value selections that allow all the COA's to be viewed is performed as follows (with the Blue Knowledge Base displayed):

| Value Selected: | New COA: |
|---|---|
| Relative Combat Power/Deception/Favors Red | BCOA3 |
| Relative Combat Power/Terrain/Equal | |
| Relative Combat Power/Mobility/Favors Red | |
| Relative Combat Power/Maneuver Units/Favors Blue | BCOA2 |
| OPFOR/OPFOR C2/Good | |
| OPFOR/Condition of OPFOR Divisions/Average | BCOA1 |
| OPFOR/Strength of OPFOR Divisions/7Ø-79% | |
| Friendly Force/Condition of Friendly Divisions/Average | RCOA2 |
| Relative Combat Power/Deception/Favors Blue | RCOA1, RCOA3 |

## APPENDIX V:
## INTACVAL SOURCE CODE

```
#include "fcntl.h"
#include "io.h"
#include "string.h"
#include "stdio.h"
#include "windows.h"
#include "process.h"
#include "dos.h"

#define MC_CLOCK                0x1000
#define MC_NOTEPAD              0x1001
#define MC_CALCULATOR           0x1002
#define MC_NEW_SESSION          0x1003
#define MC_PREVIOUS_SESSION     0x1004
#define MC_EXIT                 0x1005
#define MC_MAP                  0x1006

#define DLG_START               0x1100
#define DLG_CONTINUE            0x1101
#define DLG_FINISH              0x1102
#define DLG_FILELIST            0x1103
#define DLG_OPEN                0x1104
#define DLG_DELETE              0x1105
#define DLG_CLICK               0x1106
#define DLG_YES                 0x1107
#define DLG_NO                  0x1108
#define DLG_MISSION             0x1109
#define DLG_TERRAIN             0x110a
#define DLG_F_CAP               0x110b
#define DLG_E_CAP               0x110c
#define DLG_E_COA               0x110d
#define DLG_F_COA               0x110e
#define DLG_BUTTON1             0x110f
#define DLG_BUTTON2             0x1110
#define DLG_BUTTON              0x1111

#define ASS_FILE_NAME           "ASSUME"
#define INT_FILE_NAME           "INTENT"
#define OBJ_FILE_NAME           "OBJECTIV"
#define GEN_FILE_NAME           "GENTERRA"
#define KEY_FILE_NAME           "KEYTERRA"
#define FCOM_FILE_NAME           "FCLD"
#define FREI_FILE_NAME           "FREINF"
#define FCON_FILE_NAME           "FCOND"
#define ECOM_FILE_NAME           "ECLD"
#define EREI_FILE_NAME           "EREINF"
#define ECON_FILE_NAME           "ECOND"
#define ENEMY_FILE_NAME         "RCOA"
#define FRIENDLY_FILE_NAME      "BCOA"
#define SENEMY_FILE_NAME         "RCOAS"
#define SFRIENDLY_FILE_NAME      "BCOAS"

#define MAXLENGTH 450

typedef HANDLE HDLG;

typedef struct
```

```
 {
   char res1[21];
   char attribute;
   struct
    {
      unsigned second:5;
      unsigned minute:6;
      unsigned hour:5;
    } time;
   struct
    {
      unsigned day:5;
      unsigned month:4;
      unsigned year:7;
    } date;
   long size;
   char name[13];
 } DIR_ENTRY;

HWND main_window;
HANDLE intacval_instance;
TEXTMETRIC font_sizing;
char id_name[8];
HDLG current_figure;                        /*Handle to the current dialog box*/
FARPROC current_dialog_func;                /*The current dialog function-
                                              There is one for each source
                                              which creates dialog boxes*/
HANDLE list_handle;                         /*handle to list of session files
                                              in directory*/
char filenames[1Ø][6Ø];                     /*files used as session files and
                                              presently found in the directory
                                              including date and time*/

int new_fp;                                 /*file pointer to file opened
                                              before the CreateWindow*/
HWND file_handle[3];                        /*the current handles to open file
                                              or OAV windows*/
int no_of_open_files;                       /*number of open file_windows*/
int current_active_window;
char s[2ØØ];                                /*used as string in various
                                              functions especially in debug*/
int figflag;                                /*the current status */

/*E-Enemy Coa file for current COA*/
/*F-Friendly Coa file for current COA*/
/*EO-Enemy OAV'S*/
/*FO-Friendly OAV'S*/

                 /*FIG FLAG Table*/
/*    VALUE      RESOURCE       COMMENTS
        3Ø       Figure 3       Previous session
        4Ø       Figure 4
        45       Figure 4A      After asking in figure 4 to delete a file
        5Ø       Figure 5       After clicking START
        6Ø       Figure 5       Mission
        9Ø       Figure 5       Terrain
       11Ø       Figure 5       Friendly Cap
       12Ø      Figure 5        Enemy Cap
       13Ø      Figure 13       E
       131      Figure 13       F
       132      Figure 13       EO
       135      Figure 14       E-F
       136      Figure 14       E-EO
       137      Figure 14       F-EO
       138      Figure 5        E-F-EO
       14Ø      Figure 13       F
       141      Figure 13       E
       142      Figure 13       FO
       145      Figure 14       F-E
       146      Figure 14       F-FO
       147      Figure 14       E-FO
       148      Figure 5        F-E-FO

*/

unsigned char f_curvec[], fcoa[][3];
unsigned char e_curvec[], ecoa[][3];

unsigned int screen_height;
       /*The current COA to be displayed calculated using the curvecs*/
int current_e_coa, current_f_coa;
int current_highlight;                  /*The current button to be darkened
                                          on the left side of screen.*/
int destroyed_flag;                     /*Is window already closed*/
char current_session[15];
char next_session[15];
```

```
/* Parallax Graphics header file*/

/***********************************************/
/* Parallax Graphics definition of commands */
/*      AUTHOR: Dave Arnold     DATE: 6/17/85    */
/***********************************************/

/* (GRAPHIO.C) Subroutine: */
/*      GRAPHIO(index ,arg1,..,argn,wcnt,buffer)        */

/* opcode structure definition: tells graphio how to deal with calls */

typeaef struct pstruct
{
        int opcode;                             /* graphics opcode */
        int nparms;                     /* number of parameters including opcode
 */
        int flag; /* flag Ø = nobuffer, 1 = write buffer, -1 = read buffer */
} PSTRUCT;

/***********************************************/
/* WRITE FROM BUFFER TO GRAPHICS CONTROLLER      */
/* obuf(wcnt, buffer) int wcnt, *buffer;         */
/***********************************************/

#define OBUFF Ø         /* index of op code structure */
#define obuff(wcnt, buffer) graphio(OBUFF, wcnt, buffer)

/***********************************************/
/* READ INTO BUFFER FROM GRAPHICS CONTROLLER     */
/* ibuff(wcnt, buffer) int wcnt, *buffer;        */
/***********************************************/

#define IBUFF 1
#define ibuff(wcnt, buffer) graphio(IBUFF, wcnt, buffer)

/*****************************/
/* DRAW SOLID FILLED POLYGON */
/* poly(color,n,buffer) int color,n,*buffer; */
/*****************************/

#define POLY 2
#define poly(color,n,buffer) graphio(POLY, color, n, n << 1, buffer)

/************************/
/* DRAW OUTLINED POLYGON */
/* polyo(color,n,buffer) int color,n,*buffer; */
/************************/

#define POLYO 3
#define polyo(color,n,buffer) graphio(POLYO, color, n, n << 1, buffer)

/************************/
/* DRAW STIPPLED POLYGON */
/* polys(xstip,ystip,n,buffer) int xstip,ystip,n,*buffer; */
/************************/
```

```
#define POLYS 4
#define polys(xstip,ystip,n,buffer) graphio(POLYS,xstip,ystip,n,n << 1,buffer)

/****************/
/* COPY POLYGON */
/* polyc(xold,yold,n,buffer) int xold,yold,n,*buffer; */
/****************/

#define POLYC 5
#define polyc(xold,yold,n,buffer) graphio(POLYC, xold, yold, n, n << 1, buffer)

/**********************/
/* DRAW SOLID CIRCLE */
/* circ(color,radius,x,y) int color,radius,x,y; */
/**********************/

#define CIRC 6
#define circ(color,radius,x,y) graphio(CIRC, color, radius, x, y)

/*************************/
/* DRAW OUTLINED CIRCLE */
/* circo(color,radius,x,y) int color,radius,x,y; */
/*************************/

#define CIRCO 7
#define circo(color,radius,x,y) graphio(CIRCO, color, radius, x, y)

/*************************/
/* DRAW STIPPLED CIRCLE */
/* circs(xstip,ystip,radius,x,y) int xstip,ystip,radius,x,y; */
/*************************/

#define CIRCS 8
#define circs(xstip,ystip,radius,x,y) graphio(CIRCS,xstip,ystip,radius,x,y)

/***************/
/* COPY CIRCLE */
/* circc(xold,yold,radius,xnew,ynew) int xold,yold,radius,xnew,ynew; */
/***************/

#define CIRCC 9
#define circc(xold,yold,radius,xnew,ynew)\
graphio(CIRCC,xold,yold,radius,xnew,ynew)

/*****************/
/* INTERIOR FILL */
/* fill(color,x,y) int color,x,y; */
/*****************/

#define FILL 1Ø
#define fill(color,x,y) graphio(FILL, color, x, y)

/*************************/
/* INTERIOR STIPPLE FILL */
/* fills(xstip,ystip,x,y) int xstip,ystip,x,y; */
```

```
/*************************/

#define FILLS 11
#define fills(xstip,ystip,x,y) graphio(FILLS, xstip, ystip, x, y)

/*****************/
/* INTERIOR COPY */
/* fillc(xold,yold,xnew,ynew) int xold,yold,xnew,ynew; */
/*****************/

#define FILLC 12
#define fillc(xold,yold,xnew,ynew) graphio(FILLC, xold, yold, xnew, ynew)

/******************/
/* DRAW SOLID BOX */
/* box(color,x1,y1,x2,y2) int color,x1,y1,x2,y2; */
/******************/

#define BOX 13
#define box(color,x1,y1,x2,y2) graphio(BOX, color, x1, y1, x2, y2)

/*********************/
/* DRAW OUTLINED BOX */
/* boxo(color,x1,y1,x2,y2) int color,x1,y1,x2,y2; */
/*********************/

#define BOXO 14
#define boxo(color,x1,y1,x2,y2) graphio(BOXO,color,x1,y1,x2,y2)

/*********************/
/* DRAW STIPPLED BOX */
/* boxs(xstip,ystip,x1,y1,x2,y2) int xstip,ystip,x1,y1,x2,y2; */
/*********************/

#define BOXS 15
#define boxs(xstip,ystip,x1,y1,x2,y2) graphio(BOXS,xstip,ystip,x1,y1,x2,y2)

/************/
/* COPY BOX */
/* boxc(xold,yold,x1,y1,x2,y2) int xold,yold,x1,y1,x2,y2; */
/************/

#define BOXC 16
#define boxc(xold,yold,x1,y1,x2,y2) graphio(BOXC,xold,yold,x1,y1,x2,y2)

/**************/
/* WRITE TEXT */
/* text(color,x,y,string) int color,x,y; char *string; */
/**************/

#define TEXT 17
#define text(color,x,y,string)\
        graphio(TEXT,color,x,y,strlen(string),(strlen(string)+1) >> 1,string)

/**********************/
/* WRITE STIPPLED TEXT */
/* texts(xstip,ystip,x,y,string) int xstip,ystip,x,y; char *string; */
/*************************/

#define TEXTS 18
#define texts(xstip,ystip,x,y,string)\
graphio(TEXTS,xstip,ystip,x,y,strlen(string),(strlen(string)+1) >> 1, string)

/*************/
/* COPY TEXT */
/* textc(xold,yold,xnew,ynew,string) int xold,yold,xnew,ynew; char *string; */
/*************/

#define TEXTC 19
#define textc(xold,yold,xnew,ynew,string)\
graphio(TEXTC,xold,yold,xnew,ynew,strlen(string),\
(strlen(string)+1) >> 1, string)

/***************/
/* DRAW VECTOR */
/* vect(color,x1,y1,x2,y2) int color,x1,y1,x2,y2; */
/***************/

#define VECT 20
#define vect(color,x1,y1,x2,y2) graphio(VECT,color,x1,y1,x2,y2)

/*************************/
/* DRAW MULTIPLE VECTORS */
/* vectm(color,n,buffer) int color,n,*buffer; */
/*************************/

#define VECTM 21
#define vectm(color,n,buffer) graphio(VECTM,color,n,n << 1,buffer)

/***************/
/* VECTOR SAVE */
/* vects(vx1,vy1,vx2,vy2,bx1,by1,bx2,by2) int vx1,vy1,vx2,vy2,bx1,by1,bx2,by2; *
/
/***************/

#define VECTS 22
#define vects(vx1,vy1,vx2,vy2,bx1,by1,bx2,by2)\
graphio(VECTS,vx1,vy1,vx2,vy2,bx1,by1,bx2,by2)

/******************/
/* VECTOR RESTORE */
/* vectr(bx1,by1,bx2,by2,vx1,vy1,vx2,vy2) int bx1,by1,bx2,by2,vx1,vy1,vx2,vy2; *
/
/******************/

#define VECTR 23
#define vectr(bx1,by1,bx2,by2,vx1,vy1,vx2,vy2)\
graphio(VECTR,bx1,by1,bx2,by2,vx1,vy1,vx2,vy2)

/*******************/
/* PATTERNED VECTOR */
/* vectp(color1,color2,pattern,x1,y1,x2,y2) int
```

```
color1,color2,pattern,x1,y1,x2,y2; */
/********************/

#define VECTP 24
#define vectp(color1,color2,pattern,x1,y1,x2,y2)\
graphio(VECTP,color1,color2,pattern,x1,y1,x2,y2)

/**************/
/* XOR VECTOR */
/* vectx(value,x1,y1,x2,y2) int value,x1,y1,x2,y2; */
/**************/

#define VECTX 25
#define vectx(value,x1,y1,x2,y2) graphio(VECTX,value,x1,y1,x2,y2)

#define VECTW 26
#define vectw(width, value,x1,y1,x2,y2) graphio(VECTW,width,value,x1,y1,x2,y2)

/********************/
/* SET ZOOM FACTORS */
/* zoom(zoomx,zoomy) int zoomx,zoomy; */
/********************/

#define ZOOM 27
#define zoom(zoomx,zoomy) graphio(ZOOM,(zoomx << 8) | zoomy)

/*****************/
/* SET PAN ORIGIN */
/* pan(xleft,ytop) int xleft,ytop; */
/*****************/

#define PAN 28
#define pan(xleft,ytop) graphio(PAN,xleft,ytop)

/*****************/
/* SET COLOR TABLE */
/* clt4(color,red,green,blue) int color,red,green,blue; */
/*****************/

#define CLT4 29
#define clt4(color,red,green,blue)\
graphio(CLT4,color,(red<<1)|(green<<6)|(blue<<11))

/***************************/
/* SET 8-PLANE COLOR TABLE */
/* clt8(color,red,green,blue) int color,red,green,blue; */
/***************************/

#define CLT8 30
#define clt8(color,red,green,blue) graphio(CLT8,color|(red<<8),green|(blue<<8))

/********************/
/* color table read */
/* cltrd(color,pntr) int color,*pntr; */
/********************/
```

```
#define CLTRD 31
#define cltrd(color,pntr) graphio(CLTRD,color,2,pntr)

/*********/
/* flash */
/* flash(vx,vy,dxØ,dyØ,dx1,dy1) int vx,vy,dxØ,dyØ,dx1,dy1; */
/*********/

#define FLASH 32
#define flash(vx,vy,dxØ,dyØ,dx1,dy1) graphio(FLASH,vx,vy,dxØ,dyØ,dx1,dy1)

/**********/
/* fieldf */
/* fieldf(vx,vy,dxØ,dyØ,dx1,dy1) int vx,vy,dxØ,dyØ,dx1,dy1; */
/**********/

#define FIELDF 33
#define fieldf(vx,vy,dxØ,dyØ,dx1,dy1) graphio(FIELDF,vx,vy,dxØ,dyØ,dx1,dy1)

/*************/
/* dthron    */
/* dthron()  */
/*************/

#define DTHRON 34
#define dthron() graphio(DTHRON)

/**************/
/* dthroff    */
/* dthroff()  */
/**************/

#define DTHROFF 35
#define dthroff() graphio(DTHROFF)

/*************/
/* keyon     */
/* keyon()   */
/*************/

#define KEYON 36
#define keyon() graphio(KEYON)

/*************/
/* keyoff    */
/* keyoff()  */
/*************/

#define KEYOFF 37
#define keyoff() graphio(KEYOFF)

/*****************/
/* SET WRITE MASK */
/* mask(planes) int planes; */
/*****************/
```

```
#define MASK 38
#define mask(planes) graphio(MASK,planes)

/***********************/
/* SELECT OPAQUE TABLE */
/* opaq(n) int n;      */
/***********************/

#define OPAQ 39
#define opaq(n) graphio(OPAQ,n)

/*********************/
/* LOAD OPAQUE TABLE */
/* opaql(first,n,buffer) unsigned first,n,*buffer; */
/*********************/

#define OPAQL 4Ø
#define opaql(first,n,buffer) graphio(OPAQL,(first<<8)|(n&255),((n-1)>>4)+1,buff
er)

/*********************/
/* MASK OPAQUE TABLE */
/* opaqm(planes,value) int planes,value; */
/*********************/

#define OPAQM 41
#define opaqm(planes,value) graphio(OPAQM,(planes<<8) | value)

/*********************************/
/* WAIT FOR VERTICAL SYNC (6ØHz) */
/* sync()                                                  */
/*********************************/

#define SYNC 42
#define sync() graphio(SYNC)

/******************************/
/* CLEAR VERTICAL SYNC COUNTER */
/* syncc()                                     */
/******************************/

#define SYNCC 43
#define syncc() graphio(SYNCC)

/** -*************************/
/* LOAD VERTICAL SYNC COUNTER */
/* syncl(count) int count;       */
/*****************************/

#define SYNCL 44
#define syncl(count) graphio(SYNCL,count)

/*****************************/
/* READ VERTICAL SYNC COUNTER */
/* int syncr()                   */
/*****************************/
```

```
#define SYNCR 45
#define syncr() graphio(SYNCR,1,Ø)

/**********************/
/* SHOW CONFIGURATION */
/* show()                    */
/**********************/

#define SHOW 46
#define show() graphio(SHOW,1,Ø)

/***************/
/* READ STATUS */
/* stat()         */
/***************/

#define STAT 47
#define stat() graphio(STAT,1,Ø)

/********************/
/* FIND OPAQUE PIXEL */
/* find(y,x1,x2) int y,x1,x2; */
/********************/

#define FIND 48
#define find(y,x1,x2) graphio(FIND,y,x1,x2,1,Ø)

/************************/
/* LOAD/ENABLE CLIPPING */
/* clip(x1,y1,x2,y2) int x1,y1,x2,y2; */
/************************/

#define CLIP 49
#define clip(x1,y1,x2,y2) graphio(CLIP,x1,y1,x2,y2)

/*************************/
/* LOAD/DISABLE CLIPPING */
/* clipl(x1,y1,x2,y2) int x1,y1,x2,y2; */
/*************************/

#define CLIPL 5Ø
#define clipl(x1,y1,x2,y2) graphio(CLIPL,x1,y1,x2,y2)

/******************/
/* ENABLE CLIPPING */
/* clipe()             */
/******************/

#define CLIPE 51
#define clipe() graphio(CLIPE)

/*******************/
/* DISABLE CLIPPING */
/* clipd()                   */
/*******************/
```

```
#define CLIPD 52
#define clipd() graphio(CLIPD)

/**************************/
/* SET (READ) TRANSLATION */
/* tranr(x,y) int x,y;    */
/**************************/

#define TRANR 53
#define tranr(x,y) graphio(TRANR,x,y)

/***************************/
/* SET (WRITE) TRANSLATION */
/* tranw(x,y) int x,y;     */
/***************************/

#define TRANW 54
#define tranw(x,y) graphio(TRANW,x,y)

/****************/
/* DEJAG VECTOR */
/* djag(back,fore,x1,y1,x2,y2) int back,fore,x1,y1,x2,y2; */
/****************/

#define DJAG 55
#define djag(back,fore,x1,y1,x2,y2) graphio(DJAG,(back<<8)|fore,x1,y1,x2,y2)

/*****************/
/* DEJAG (RIGHT) */
/* djagr(back,fore,x1,y1,x2,y2) int back,fore,x1,y1,x2,y2; */
/*****************/

#define DJAGR 56
#define djagr(back,fore,x1,y1,x2,y2) graphio(DJAGR,(back<<8)|fore,x1,y1,x2,y2)

/****************/
/* DEJAG (LEFT) */
/* djagl(back,fore,x1,y1,x2,y2) int back,fore,x1,y1,x2,y2; */
/****************/

#define DJAGL 57
#define djagl(back,fore,x1,y1,x2,y2) graphio(DJAGL,(back<<8)|fore,x1,y1,x2,y2)

/**********/
/* SCREEN  Ø - 64Ø mode    1 - 512 mode */
/* screen(size) int size; */
/**********/

#define SCREEN 58
#define screen(size) graphio(SCREEN,Ø25+((size<<8)&255))

/***************
* DISPLAY MODE *                   all this must change!!!!!!!!!!
***************
disp(n)
```

```
int n;
{
        if (n == 1) ORPG(1,Øx1Ø15,Ø);
        else if (n == 2) ORPG(1,Øx1115,Ø);
        else if (n == 3) ORPG(1,Øx1215,Ø);
} */
/****************/
/* NO-OPERATION */
/* noop()          */
/****************/

#define NOOP 59
#define noop() graphio(NOOP)

/************************/
/* LOAD IMAGE IMMEDIATE */
/* limgi(x1,y1,x2,y2,buffer,n) int x1,y1,x2,y2,*buffer,n; */
/************************/

#define LIMGI 6Ø
#define limgi(x1,y1,x2,y2,buffer,n) graphio(LIMGI,x1,y1,x2,y2,n,buffer)

/**************************/
/* UNLOAD IMAGE IMMEDIATE */
/* uimgi(x1,y1,x2,y2,buffer,n) int x1,y1,x2,y2,*buffer,n; */
/**************************/

#define UIMGI 61
#define uimgi(x1,y1,x2,y2,buffer,n) graphio(UIMGI,x1,y1,x2,y2,n,buffer)

/**********************************/
/* LOAD RUNLENGTH IMAGE IMMEDIATE */
/* lruni(x1,y1,x2,y2,buffer,n) int x1,y1,x2,y2,*buffer,n; */
/**********************************/

#define LRUNI 62
#define lruni(x1,y1,x2,y2,buffer,n) graphio(LRUNI,x1,y1,x2,y2,n,buffer)

/************************************/
/* UNLOAD RUNLENGTH IMAGE IMMEDIATE */
/* uruni(x1,y1,x2,y2,buffer,n) int x1,y1,x2,y2,*buffer,n; */
/************************************/

#define URUNI 63
#define uruni(x1,y1,x2,y2,buffer,n) graphio(URUNI,x1,y1,x2,y2,n,buffer)

#define SCTEXT 64
#define sctext(t,sxsy,color,x,y,n,string)\
graphio(SCTEXT,t,sxsy,color,x,y,n,(n+1) >> 1,string)

#define EFLASH 65
#define eflash(xs,ys,xØ,yØ,x1,y1) graphio(EFLASH,xs,ys,xØ,yØ,x1,y1)

#define FBLIT 66
#define fblit(bitplane,color) graphio(FBLIT,bitplane,color)
```

```
#define DTHRC DTHRON
#define dthrc() graphio(DTHRON)

#define DAMVG 67
#define damvg() graphio(DAMVG)

#define DAMVX 68
#define damvx() graphio(DAMVX)

#define DAMGG 69
#define damgg() graphio(DAMGG)

#define DAMVV 7Ø
#define damvv() graphio(DAMVV)

#define EGOV 71
#define egov() graphio(EGOV)

#define DGOV 72
#define dgov() graphio(DGOV)

#define DTHRL 73
#define dthrl() graphio(DTHRL)

#define DTHRLC 74
#define dthrlc() graphio(DTHRLC)

#define BOXSQ 75
#define boxsq(xsØ,ysØ,xs1,ys1,xdØ,ydØ,xd1,yd1)\
graphio(BOXSQ,xsØ,ysØ,xs1,ys1,xdØ,ydØ,xd1,yd1)

#define RMAP  76
#define rmap(n) graphio(RMAP, n)

#define RMAPL 77
#define rmapl(start, count, string)\
graphio(RMAPL, ((start) << 8) ! ((count) & 255), ((count) + 1)/2, string)

#define STIP8 78
#define stip8() graphio(STIP8)

#define STIP16 79
#define stip16() graphio(STIP16)

#define BOXZV 8Ø
#define boxzv(sxØ,syØ,sx1,sy1,dxØ,dyØ,dx1,dy1)\
graphio(BOXZV,sxØ,syØ,sx1,sy1,dxØ,dyØ,dx1,dy1)
```

```
#define BORDER      15
#define INLEFT1    14Ø
#define INLEFT2    18Ø
#define INRIGHT1   46Ø
#define INRIGHT2   5ØØ
#define OUTBOT1    1ØØ
#define OUTBOT2    14Ø
#define OUTTOP1    34Ø
#define OUTTOP2    38Ø

/*
  CURSOR INFORMATION

Configurable info:
*/

#define MINX         Ø
#define MINY         Ø
#define MAXX       639
#define MAXY       479
```

```
/*************   This is lvm.h */

#include "gwindows.h"

/*
  Cursor types
*/

#define XHCUR  Ø      /* crosshair */
#define SWCUR  1      /* southwest */
#define SOCUR  2      /* south     */
#define SECUR  3      /* southeast */
#define WECUR  4      /* west      */
#define EACUR  5      /* east      */
#define NWCUR  6      /* northwest */
#define NOCUR  7      /* north     */
#define NECUR  8      /* northeast */
#define CRCUR  9      /* circle    */

#define MAXTRACK 127          /* maximum number of mission tracks in mem */

/*
  parameters for display of various icons
*/


/*  parameters for display of various icons */

#define ICON_SAM                  Ø
#define ICON_RADIO                1
#define ICON_RADAR                2
#define ICON_POST_ENEMY           3

/* icon class values and definitions */

#define BRIGADE                 Ø
#define CORPS                   1
#define HQ                      2
#define REGIMENT                3
#define DIVISION                4
#define COA                     5
#define TERRAIN                 6

#define BLUE_UNIT               Ø
#define RED_UNIT                8

typedef struct
   {
     float      x, y;
   } FPOINT;

typedef struct
  {
    long lat;           /* latitude of the icon, and primary sort key */
    long lon;           /* longitude of the icon, and secondary sort key */
    unsigned class;     /* classification of type of icon */
```

```
 } icon_struct;         /* icon description structure */

extern int  markx, marky;       /* zoom mark location (pixel coordinates) */
extern long mark_lat, mark_lon;

extern int  show_mask;          /* mask of icons/tracks to be shown */
extern icon_struct *icons;
extern IMAGE *icontab;        /* table pointing to graphics icons in memory */

/* icons are displayed in area of interest bounded by the lat/longs of
   the following variables */

extern long aoi_lat1, aoi_lon1, aoi_lat2, aoi_lon2;
extern int  aoi_x1, aoi_y1, aoi_x2, aoi_y2;

pix_to_latlon(long, long, long *, long *);
latlon_to_pix(long, long, long *, long *);
long split_deg(long);
long merge_deg(long);
```

```
#ifndef PASCAL
#define PASCAL  pascal
#endif

#define EOR    '\22'
#define FALSE   Ø
#define TRUE    1
#define NULL    Ø

#define FAR     far
#define NEAR    near
#define LONG    long
#define VOID    void

typedef unsigned char   BYTE;
typedef unsigned short  WORD;
typedef unsigned long  DWORD;
typedef int         BOOL;
typedef char      *PSTR;
typedef char NEAR*NPSTR;
typedef char FAR *LPSTR;
typedef int  FAR *LPINT;
```

```
ATOM          FAR PASCAL AddAtom( LPSTR );
ATOM          FAR PASCAL DeleteAtom( ATOM );
ATOM          FAR PASCAL FindAtom( LPSTR );
WORD          FAR PASCAL GetAtomName( ATOM, LPSTR, int  );
HANDLE        FAR PASCAL GetAtomHandle( ATOM );
#define MAKEINTATOM(i)  (LPSTR)((DWORD)((WORD)i))
#endif

/* Interface to the user profile */

int           FAR PASCAL GetProfileInt( LPSTR, LPSTR, int );
int           FAR PASCAL GetProfileString( LPSTR, LPSTR, LPSTR, LPSTR, int );
BOOL          FAR PASCAL WriteProfileString( LPSTR, LPSTR, LPSTR );

/* Interface to FatalExit procedure */

void          FAR PASCAL FatalExit( int );

/* Interface to Catch and Throw procedures */

typedef int CATCHBUF[ 9 ];
typedef int FAR *LPCATCHBUF;
int           FAR PASCAL Catch( LPCATCHBUF );
void          FAR PASCAL Throw( LPCATCHBUF, int );

HANDLE        FAR PASCAL CreateMetaFile(LPSTR);
HANDLE        FAR PASCAL CloseMetaFile(HANDLE);
HANDLE        FAR PASCAL GetMetaFileBits(HANDLE);
HANDLE        FAR PASCAL SetMetaFileBits(HANDLE);

long          FAR PASCAL GetCurrentTime(void);
BOOL          FAR PASCAL IsChild(HWND, HWND);

#ifndef NOWINOFFSETS
WORD          FAR PASCAL GetWindowWord(HWND, int);
WORD          FAR PASCAL SetWindowWord(HWND, int, WORD);
LONG          FAR PASCAL GetWindowLong(HWND, int);
LONG          FAR PASCAL SetWindowLong(HWND, int, LONG);
WORD          FAR PASCAL GetClassWord(HWND, int);
WORD          FAR PASCAL SetClassWord(HWND, int, WORD);
LONG          FAR PASCAL GetClassLong(HWND, int);
LONG          FAR PASCAL SetClassLong(HWND, int, LONG);
#endif

HWND          FAR PASCAL GetParent(HWND);
BOOL          FAR PASCAL EnumChildWindows(HWND, FARPROC, LONG);
HWND          FAR PASCAL FindWindow(LPSTR, LPSTR);
BOOL          FAR PASCAL EnumWindows(FARPROC, LONG);
int           FAR PASCAL GetClassName(HWND, LPSTR, int);

#ifndef NOWH
FARPROC       FAR PASCAL SetWindowsHook(int, FARPROC);
#endif

/* Key conversion window */
HWND          FAR PASCAL CreateConvertWindow( LPSTR, HANDLE, LPSTR );
```

```
void        FAR PASCAL ShowConvertWindow( HWND, BOOL );
void        FAR PASCAL SetConvertWindowHeight( int );
BOOL        FAR PASCAL IsTwoByteCharPrefix( char );
#ifndef NOMENUS

/* Menu flags for Add/Check/EnableMenuItem */
#define MF_CHANGE       0x0080
#define MF_INSERT       0x0000
#define MF_APPEND       0x0100
#define MF_DELETE       0x0200
#define MF_BYPOSITION   0x0400
#define MF_SEPARATOR    0x0800
#define MF_BYCOMMAND    0x0000
#define MF_GRAYED       0x0001
#define MF_DISABLED     0x0002
#define MF_ENABLED      0x0000
#define MF_CHECKED      0x0008
#define MF_UNCHECKED    0x0000
#define MF_BITMAP       0x0004
#define MF_STRING       0x0000
#define MF_POPUP        0x0010
#define MF_MENUBARBREAK 0x0020
#define MF_MENUBREAK    0x0040
#define MF_HILITE       0x0080
#define MF_UNHILITE     0x0000

#endif /* of NOMENU */

/* System Menu Command Values */
#ifndef NOSYSCOMMANDS
#define SC_SIZE         0xf000
#define SC_MOVE         0xf010
#define SC_ICON         0xf020
#define SC_ZOOM         0xf030
#define SC_NEXTWINDOW   0xf040
#define SC_PREVWINDOW   0xf050
#define SC_CLOSE        0xf060
#define SC_VSCROLL      0xf070
#define SC_HSCROLL      0xf080
#define SC_MOUSEMENU    0xf090
#define SC_KEYMENU      0xf100
#endif

/* Resource loading routines */
#ifndef NOBITMAP
HBITMAP     FAR PASCAL LoadBitmap( HANDLE, LPSTR );
#endif

HCURSOR     FAR PASCAL LoadCursor( HANDLE, LPSTR );

/* Standard cursor IDs */
#define IDC_ARROW       MAKEINTRESOURCE(32512)
#define IDC_IBEAM       MAKEINTRESOURCE(32513)
#define IDC_WAIT        MAKEINTRESOURCE(32514)
#define IDC_CROSS       MAKEINTRESOURCE(32515)
#define IDC_UPARROW     MAKEINTRESOURCE(32516)
```

```
#define IDC_SIZE        MAKEINTRESOURCE(3264Ø)
#define IDC_ICON        MAKEINTRESOURCE(32641)


HICON       FAR PASCAL LoadIcon( HANDLE, LPSTR );

#ifndef NOICON
/* Standard icon IDs */
#define IDI_APPLICATION MAKEINTRESOURCE(32512)
#define IDI_HAND        MAKEINTRESOURCE(32513)
#define IDI_QUESTION    MAKEINTRESOURCE(32514)
#define IDI_EXCLAMATION MAKEINTRESOURCE(32515)
#define IDI_ASTERISK    MAKEINTRESOURCE(32516)
#endif


#ifndef NOMENUS
HMENU       FAR PASCAL LoadMenu( HANDLE, LPSTR );
#endif

int         FAR PASCAL LoadString( HANDLE, unsigned, LPSTR, int );

short       FAR PASCAL AddFontResource( LPSTR );
BOOL        FAR PASCAL RemoveFontResource( LPSTR );

#ifndef NOKANJI
#define CP_HWND                 Ø
#define CP_OPEN                 1
#define CP_DIRECT               2

typedef struct{
    short   x;
    short   y;
    LPSTR   lpYomi;
    LPSTR   lpResult;
    short   YomiCount;
    short   ResultCount;
} KANJISTRUCT;

typedef KANJISTRUCT  FAR *LPKANJISTRUCT;

VOID        FAR PASCAL  MoveConvertWindow (short, short);
VOID        FAR PASCAL  ConvertRequest (HWND, LPKANJISTRUCT);
BOOL        FAR PASCAL  SetConvertParams(short, short);
VOID        FAR PASCAL  SetConvertHook(BOOL);
#endif

/* Conventional dialog box and message box command IDs */
#define IDOK            1
#define IDCANCEL        2
#define IDABORT         3
#define IDRETRY         4
#define IDIGNORE        5
#define IDYES           6
#define IDNO            7
```

```
#ifndef NOCTLMGR

/* Control manager structures & definitions */
/* Edit control class stuff */

/* styles */
#ifndef NOWINSTYLES
#define ES_LEFT             0L
#define ES_CENTER           1L
#define ES_RIGHT            2L
#define ES_MULTILINE        4L
#define ES_AUTOVSCROLL      64L
#define ES_AUTOHSCROLL      128L
#define ES_NOHIDESEL        256L
#endif

/* notification codes */
#define EN_SETFOCUS     0x0100
#define EN_KILLFOCUS    0x0200
#define EN_CHANGE       0x0300
#define EN_ERRSPACE     0x0500
#define EN_HSCROLL      0x0601
#define EN_VSCROLL      0x0602

/* control messages: */
#ifndef NOWINMESSAGES
#define EM_GETSEL           WM_USER+0
#define EM_SETSEL           WM_USER+1
#define EM_GETRECT          WM_USER+2
#define EM_SETRECT          WM_USER+3
#define EM_SETRECTNP        WM_USER+4
#define EM_SCROLL           WM_USER+5
#define EM_LINESCROLL       WM_USER+6
#define EM_GETMODIFY        WM_USER+8
#define EM_SETMODIFY        WM_USER+9
#define EM_GETLINECOUNT     WM_USER+10
#define EM_LINEINDEX        WM_USER+11
#define EM_SETHANDLE        WM_USER+12
#define EM_GETHANDLE        WM_USER+13
#define EM_GETTHUMB         WM_USER+14
#define EM_LINELENGTH       WM_USER+17
#define EM_REPLACESEL       WM_USER+18
#define EM_SETFONT          WM_USER+19
#define EM_GETLINE          WM_USER+20
#define EM_LIMITTEXT        WM_USER+21
#define EM_CANUNDO          WM_USER+22
#define EM_UNDO             WM_USER+23
#define EM_FMTLINES         WM_USER+24
#endif

/* button control styles */
#define BS_PUSHBUTTON       0L
#define BS_DEFPUSHBUTTON    1L
#define BS_CHECKBOX         2L
#define BS_AUTOCHECKBOX     3L
#define BS_RADIOBUTTON      4L
```

```
#define BS_3STATE         5L
#define BS_AUTO3STATE     6L
#define BS_GROUPBOX       7L
#define BS_USERBUTTON     8L

/* user button notification codes */
#define BN_CLICKED        0
#define BN_PAINT          1
#define BN_HILITE         2
#define BN_UNHILITE       3
#define BN_DISABLE        4

/* control messages */
#define BM_GETCHECK      WM_USER+0
#define BM_SETCHECK      WM_USER+1
#define BM_GETSTATE      WM_USER+2
#define BM_SETSTATE      WM_USER+3

/* Static control constants */

#define SS_LEFT        0L
#define SS_CENTER      1L
#define SS_RIGHT       2L
#define SS_ICON        3L
#define SS_BLACKRECT   4L
#define SS_GRAYRECT    5L
#define SS_WHITERECT   6L
#define SS_BLACKFRAME  7L
#define SS_GRAYFRAME   8L
#define SS_WHITEFRAME  9L
#define SS_USERITEM    10L

/* Dialog manager routines */

#ifndef NOMSG
BOOL        FAR PASCAL IsDialogMessage(HWND, LPMSG);
#endif

#ifndef NORECT
void        FAR PASCAL MapDialogRect(HWND, LPRECT);
#endif

#ifndef NOCTLMGR
int         FAR PASCAL DlgDirList(HWND, LPSTR, int, int, unsigned);
BOOL        FAR PASCAL DlgDirSelect(HWND, LPSTR, int);

/* Dialog style bits */
#define DS_ABSALIGN    0x000000001L
#define DS_SYSMODAL    0x000000002L

#define LB_CTLCODE    0L

/* Listbox control return values */
#define LB_OKAY       0
#define LB_ERR        -1
#define LB_ERRSPACE   -2
```

```
/* listbox notification codes */
#define LBN_ERRSPACE    -2
#define LBN_SELCHANGE   1
#define LBN_DBLCLK      2
#endif

/* listbox messages */
#ifndef NOWINMESSAGES
#define LB_ADDSTRING      1+WM_USER
#define LB_INSERTSTRING  2+WM_USER
#define LB_DELETESTRING  3+WM_USER
#define LB_REPLACESTRING 4+WM_USER
#define LB_RESETCONTENT  5+WM_USER
#define LB_SETSEL        6+WM_USER
#define LB_SETCURSEL     7+WM_USER
#define LB_GETSEL        8+WM_USER
#define LB_GETCURSEL     9+WM_USER
#define LB_GETTEXT       10+WM_USER
#define LB_GETTEXTLEN    11+WM_USER
#define LB_GETCOUNT      12+WM_USER
#define LB_SELECTSTRING  13+WM_USER
#define LB_DIR           14+WM_USER
#define LB_MSGMAX        15+WM_USER
#endif

/* listbox style bits */
#ifndef NOWINSTYLES
#define LBS_NOTIFY        0x0001L
#define LBS_SORT          0x0002L
#define LBS_NOREDRAW      0x0004L
#define LBS_MULTIPLESEL   0x0008L
#define LBS_STANDARD      (LBS_NOTIFY | LBS_SORT | WS_VSCROLL | WS_BORDER)
#endif

/* scroll bar styles */
#ifndef NOWINSTYLES
#define SBS_HORZ                     0x0000L
#define SBS_VERT                     0x0001L
#define SBS_TOPALIGN                 0x0002L
#define SBS_LEFTALIGN                0x0002L
#define SBS_BOTTOMALIGN              0x0004L
#define SBS_RIGHTALIGN               0x0004L
#define SBS_SIZEBOXTOPLEFTALIGN      0x0002L
#define SBS_SIZEBOXBOTTOMRIGHTALIGN 0x0004L
#define SBS_SIZEBOX                  0x0008L
#endif
#endif

#ifndef NOSOUND
int         FAR PASCAL OpenSound();
int         FAR PASCAL CloseSound();
int         FAR PASCAL SetVoiceQueueSize(int, int);
int         FAR PASCAL SetVoiceNote(int, int, int, int);
int         FAR PASCAL SetVoiceAccent(int, int, int, int, int);
int         FAR PASCAL SetVoiceEnvelope(int, int, int);
int          FAR PASCAL SetSoundNoise(int, int);
int          FAR PASCAL SetVoiceSound(int, int, int);
int          FAR PASCAL StartSound();
int          FAR PASCAL StopSound();
int          FAR PASCAL WaitSoundState(int);
int          FAR PASCAL SyncAllVoices();
int          FAR PASCAL CountVoiceNotes(int);
LPINT        FAR PASCAL GetThresholdEvent();
int          FAR PASCAL GetThresholdStatus();
int          FAR PASCAL SetVoiceThreshold(int, int);

/* constants used to specify return condition for WaitSoundState */

#define QUEUEEMPTY      Ø
#define THRESHOLD       1
#define ALLTHRESHOLD_   2

/* constants used to specify accent mode */

#define     S_NORMAL      Ø
#define     S_LEGATO      1
#define     S_STACCATO    2

/* constants used to specify source in SetSoundNoise */
#define     S_PERIOD512   Ø   /* freq = N/512 high pitch, less coarse hiss */
#define     S_PERIOD1Ø24  1   /* freq = N/1Ø24 */
#define     S_PERIOD2Ø48  2   /* freq = N/2Ø48 low pitch, more coarse hiss */
#define     S_PERIODVOICE 3   /* source is frequency from voice channel (3) */

#define     S_WHITE512    4   /* freq = N/512 high pitch, less coarse hiss */
#define     S_WHITE1Ø24   5   /* freq = N/1Ø24 */
#define     S_WHITE2Ø48   6   /* freq = N/2Ø48 low pitch, more coarse hiss */
#define     S_WHITEVOICE  7   /* source is frequency from voice channel (3) */

#define     S_SERDVNA      -1      /* device not available */
#define     S_SEROFM       -2      /* out of memory */
#define     S_SERMACT      -3      /* music active */
#define     S_SERQFUL      -4      /* queue full */
#define     S_SERBDNT      -5      /* invalid note */
#define     S_SERDLN       -6      /* invalid note length */
#define     S_SERDCC       -7      /* invalid note count */
#define     S_SERDTP       -8      /* invalid tempo */
#define     S_SERDVL       -9      /* invalid volume */
#define     S_SERDMD       -1Ø     /* invalid mode */
#define     S_SERDSH       -11     /* invalid shape */
#define     S_SERDPT       -12     /* invalid pitch */
#define     S_SERDFQ       -13     /* invalid frequency */
#define     S_SERDDR       -14     /* invalid duration */
#define     S_SERDSR       -15     /* invalid source */
#define     S_SERDST       -16     /* invalid state */
#endif


#ifndef NOCOMM
/*****************************************************************************
```

```
**
** dcb field definitions.
**
*********************************************************************/

#define NOPARITY         0
#define ODDPARITY        1
#define EVENPARITY       2
#define MARKPARITY       3
#define SPACEPARITY      4

#define ONESTOPBIT       0
#define ONE5STOPBITS     1
#define TWOSTOPBITS      2

#define IGNORE           0              /* Ignore signal                */
#define INFINITE         0xffff         /* Infinite timeout             */


/*********************************************************************
**
** Comm Device Driver Error Bits.
**
*********************************************************************/

#define CE_RXOVER        0x0001         /* Receive Queue overflow       */
#define CE_OVERRUN       0x0002         /* Receive Overrun Error        */
#define CE_RXPARITY      0x0004         /* Receive Parity Error         */
#define CE_FRAME         0x0008         /* Receive Framing error        */
#define CE_BREAK         0x0010         /* Break Detected               */
#define CE_CTSTO         0x0020         /* CTS Timeout                  */
#define CE_DSRTO         0x0040         /* DSR Timeout                  */
#define CE_RLSDTO        0x0080         /* RLSD Timeout                 */
#define CE_TXFULL        0x0100         /* TX QUEUE IS FULL             */
#define CE_PTO           0x0200         /* LPTx Timeout                 */
#define CE_IOE           0x0400         /* LPTx I/O Error               */
#define CE_DNS           0x0800         /* LPTx Device not selected     */
#define CE_OOP           0x1000         /* LPTx Out-Of-Paper            */
#define CE_MODE          0x8000         /* Requested mode unsupported   */

/*********************************************************************
**
** Initialization Error Codes
**
*********************************************************************/

#define IE_BADID         -1             /* Invalid or unsupported id    */
#define IE_OPEN          -2             /* Device Already Open          */
#define IE_NOPEN         -3             /* Device Not Open              */
#define IE_MEMORY        -4             /* Unable to allocate queues    */
#define IE_DEFAULT       -5             /* Error in default parameters  */
#define IE_HARDWARE      -10            /* Hardware Not Present         */
#define IE_BYTESIZE      -11            /* Illegal Byte Size            */
#define IE_BAUDRATE      -12            /* Unsupported BaudRate         */
```

```
/*****************************************************************
**
** Event Definitions
**
*****************************************************************/

#define EV_RXCHAR       0x0001          /* Any Character received      */
#define EV_RXFLAG       0x0002          /* Received certain character  */
#define EV_TXEMPTY      0x0004          /* Transmitt Queue Empty       */
#define EV_CTS          0x0008          /* CTS changed state           */
#define EV_DSR          0x0010          /* DSR changed state           */
#define EV_RLSD         0x0020          /* RLSD changed state          */
#define EV_BREAK        0x0040          /* BREAK received              */
#define EV_ERR          0x0080          /* Line status error occurred  */
#define EV_RING         0x0100          /* Ring signal detected        */
#define EV_PERR         0x0200          /* Printer error occured       */

/*****************************************************************
**
** Escape Functions
**
*****************************************************************/

#define SETXOFF         1               /* Simulate XOFF received      */
#define SETXON          2               /* Simulate XON received       */
#define SETRTS          3               /* Set RTS high                */
#define CLRRTS          4               /* Set RTS low                 */
#define SETDTR          5               /* Set DTR high                */
#define CLRDTR          6               /* Set DTR low                 */
#define RESETDEV        7               /* Reset device if possible    */


*****************************************************************
**
** Device Descriptor Block Definition
**
*****************************************************************/

#define LPTx    0x80                    /* Set if ID is for LPT device  */

typedef struct {
    BYTE     Id;                        /* Internal Device ID           */
    WORD     BaudRate;                  /* Baudrate at which runing     */
    BYTE     ByteSize;                  /* Number of bits/byte, 4-8     */
    BYTE     Parity;                    /* 0-4=None,Odd,Even,Mark,Space */
    BYTE     StopBits;                  /* 0,1,2 = 1, 1.5, 2            */
    WORD     RlsTimeout;                /* Timeout for RLSD to be set   */
    WORD     CtsTimeout;                /* Timeout for CTS to be set    */
    WORD     DsrTimeout;                /* Timeout for DSR to be set    */

    BYTE     fBinary: 1;                /* Binary Mode (skip EOF check  */
    BYTE     fRtsDisable:1;             /* Don't assert RTS at init time*/
    BYTE      fParity: 1;                 /* Enable parity checking       */
    BYTE      fOutxCtsFlow:1;             /* CTS handshaking on output    */
    BYTE      fOutxDsrFlow:1;             /* DSR handshaking on output    */
    BYTE      fDummy: 2;                  /* Reserved                     */
    BYTE      fDtrDisable:1;              /* Don't assert DTR at init time*/

    BYTE      fOutX: 1;                   /* Enable output X-ON/X-OFF     */
    BYTE      fInX: 1;                    /* Enable input X-ON/X-OFF      */
    BYTE      fPeChar: 1;                 /* Enable Parity Err Replacement*/
    BYTE      fNull: 1;                   /* Enable Null stripping        */
    BYTE      fChEvt: 1;                  /* Enable Rx character event.   */
    BYTE      fDtrflow: 1;                /* DTR handshake on input       */
    BYTE      fRtsflow: 1;                /* RTS handshake on input       */
    BYTE      fDummy2: 1;

    char      XonChar;                    /* Tx and Rx X-ON character     */
    char      XoffChar;                   /* Tx and Rx X-OFF character    */
    WORD      XonLim;                     /* Transmit X-ON threshold      */
    WORD      XoffLim;                    /* Transmit X-OFF threshold     */
    char      PeChar;                     /* Parity error replacement char*/
    char      EofChar;                    /* End of Input character       */
    char      EvtChar;                    /* Recieved Event character     */
    WORD      TxDelay;                    /* Amount of time between chars */
    } DCB;

/*****************************************************************
**
** Status record returned by GetCommErro.
**
*****************************************************************/

typedef struct {
    BYTE          fCtsHold: 1;            /* Transmit is on CTS hold      */
    BYTE          fDsrHold: 1;            /* Transmit is on DSR hold      */
    BYTE          fRlsdHold: 1;           /* Transmit is on RLSD hold     */
    BYTE          fXoffHold: 1;           /* Received handshake           */
    BYTE          fXoffSent: 1;           /* Issued handshake             */
    BYTE          fEof: 1;                /* End of file character found  */
    BYTE          fTxim: 1;               /* Character being transmitted  */
    WORD          cbInQue;                /* count of characters in Rx Que*/
    WORD          cbOutQue;               /* count of characters in Tx Que*/
    } COMSTAT;

short    FAR     PASCAL  OpenComm(LPSTR, WORD, WORD);
short    FAR     PASCAL  SetCommState(DCB FAR *);
short    FAR     PASCAL  GetCommState(short, DCB FAR *);
short    FAR     PASCAL  ReadComm(short, LPSTR, int);
short    FAR     PASCAL  UngetCommChar(short, char);
short    FAR     PASCAL  WriteComm(short, LPSTR, int);
short    FAR     PASCAL  CloseComm(short);
short    FAR     PASCAL  GetCommError(short, COMSTAT FAR *);
short    FAR     PASCAL  BuildCommDCB(LPSTR, DCB FAR *);
short    FAR     PASCAL  TransmitCommChar(short, char);
WORD FAR * FAR   PASCAL  SetCommEventMask(short, WORD);
WORD    FAR     PASCAL  GetCommEventMask(short, int);
short   FAR     PASCAL  SetCommBreak(short);
short   FAR     PASCAL  ClearCommBreak(short);
short   FAR     PASCAL  FlushComm(short, int);
short   FAR     PASCAL  EscapeCommFunction(short, int);
#endif
```

```
#include "intacval.h"

long pascal HandleFileWindow (HWND, WORD, WORD, LONG);
long pascal HandleBoxesWindow (HWND, WORD, WORD, LONG);

pascal main_dialog(hWnd, wMsg, wParam, lParam)
  HWND hWnd;
  WORD wMsg, wParam;
  LONG lParam;

{
 long li;
 int i;

 switch(wMsg)
  {case WM_COMMAND:
       switch(wParam)
        {
         case DLG_FILELIST:
          {
           switch(HIWORD(lParam))
            {
             case LBN_SELCHANGE:
              {
               if (figflag == 4Ø) return TRUE;
               li = SendDlgItemMessage(current_figure, DLG_FILELIST,
                   LB_GETCURSEL,NULL,(LONG)NULL);
               if(li != -1l)
                {
                 DestroyWindow(current_figure);
                 FreeProcInstance(current_dialog_func);
                 current_dialog_func =
                   MakeProcInstance((FARPROC)main_dialog,intacval_instance);
                 current_figure = CreateDialog(intacval_instance,
                   (LPSTR)"Figure4", main_window.current_dialog_func);
                 for(i = Ø; i< 1Ø; i++)
                  {
                   if (filenames[i][Ø] == ' ')
                     break;
                   SendDlgItemMessage(current_figure,DLG_FILELIST,
                     LB_ADDSTRING,NULL, (LONG)(LPSTR)filenames[i]);
                  }
                 SendDlgItemMessage(current_figure,DLG_FILELIST,
                   LB_SETCURSEL,(int)li,(LONG)NULL);
                }
               figflag = 4Ø;
               return TRUE;
              }
             case LBN_DBLCLK:
              {
               SendMessage(current_figure,WM_COMMAND,DLG_OPEN,
                  (LONG)(LPSTR)NULL);
               return TRUE;
              }
             default:
              return FALSE;
            }
          }
        case DLG_OPEN:
            {
             li = SendDlgItemMessage(current_figure, DLG_FILELIST,
               LB_GETCURSEL,NULL,(LONG)NULL);
             SendDlgItemMessage(current_figure,DLG_FILELIST, LB_GETTEXT,
               (int)li,(LONG)(LPSTR)s);
             open_file(s);
             figflag = Ø;
             return TRUE;
            }
        case DLG_DELETE:
            {
             li = SendDlgItemMessage(current_figure, DLG_FILELIST,
                 LB_GETCURSEL,NULL,(LONG)NULL);
             if(li != -11)
              {
               DestroyWindow(current_figure);
               FreeProcInstance(current_dialog_func);
               current_dialog_func =
                    MakeProcInstance((FARPROC)main_dialog, intacval_instance);
               current_figure = CreateDialog(intacval_instance,
                    (LPSTR)"Figure4A", main_window,current_dialog_func);
               for(i = Ø; i< 1Ø; i++)
                {
                 if (filenames[i][Ø] == ' ')
                  break;
                 SendDlgItemMessage(current_figure, DLG_FILELIST,
                    LB_ADDSTRING,NULL, (LONG)(LPSTR)filenames[i]);
                }
               SendDlgItemMessage(current_figure,DLG_FILELIST,LB_SETCURSEL,
                 (int)li,(LONG)NULL);
              }
             figflag = 45;
             return TRUE;
            }
        case DLG_NO:
            {
             DestroyWindow(current_figure);
             FreeProcInstance(current_dialog_func);
             current_dialog_func =
                  MakeProcInstance((FARPROC)main_dialog, intacval_instance);
             current_figure = CreateDialog(intacval_instance,
                  (LPSTR)"Figure3", main_window,current_dialog_func);
             for(i = Ø; i< 1Ø; i++)
              {
               if (filenames[i][Ø] == ' ')
                break;
               SendDlgItemMessage(current_figure,DLG_FILELIST, LB_ADDSTRING,
                 NULL, (LONG)(LPSTR)filenames[i]);
              }
             figflag = 3Ø;
             return TRUE;
            }
        case DLG_YES:
```

```
        {
         li = SendDlgItemMessage(current_figure,DLG_FILELIST,
            LB_GETCURSEL,NULL,(LONG)NULL);
         SendDlgItemMessage(current_figure, DLG_FILELIST, LB_GETTEXT,
            (int)li,(LONG)(LPSTR)s);
         delete_file(s);
         figflag = Ø;
         return TRUE;
        }
    case DLG_START:
        {
         destroy_current_childs().
         DestroyWindow(current_figure);
         FreeProcInstance(current_dialog_func);
         current_dialog_func =
              MakeProcInstance((FARPROC)main_dialog, intacval_instance);
         current_figure = CreateDialog(intacval_instance,
              (LPSTR)"Figure5", main_window,current_dialog_func);
         figflag = 5Ø;
         return TRUE;
        }
    case DLG_MISSION:
        {
         get_mission_files();
         figflag = 6Ø;
         return TRUE;
        }
    case DLG_TERRAIN:
        {
         get_terrain_files();
         figflag = 9Ø;
         return TRUE;
        }
    case DLG_F_CAP:
        {
         get_friendly_files();
         figflag = 11Ø;
         return TRUE;
        }
    case DLG_E_CAP:
        {
         get_enemy_files();
         figflag = 12Ø;
         return TRUE;
        }
    case DLG_E_COA:
        {
         destroy_current_childs();
         current_highlight = DLG_E_COA;
         one_window_two_buttons(wParam);
         return TRUE;
        }
    case DLG_F_COA:
        {
         destroy_current_childs();
         current_highlight = DLG_F_COA;
```

```
                one_window_two_buttons(wParam);
                return TRUE;
               }
          case DLG_CONTINUE:
               {
                continue_func();
                return TRUE;
               }
          case DLG_BUTTON1: case DLG_BUTTON2:
               {
                two_windows_one_button(wParam);
                return TRUE;
               }
          case DLG_BUTTON:
               {
                three_windows_no_buttons(wParam);
                return TRUE;
               }
          default:
               figflag = Ø;
               return FALSE;
         }
   }
  return FALSE;
 }

long pascal HandleMainWindow (hWnd, wMsg, wParam, lParam)

   HWND hWnd;
   WORD wMsg, wParam;
   LONG lParam;

 {

   switch (wMsg)
    {
      case WM_SYSCOMMAND:
       switch (wParam & Øxfff0)
        {
         case SC_KEYMENU:
          if (lParam == 9)
           {
            return(DefWindowProc (hWnd, wMsg, SC_NEXTWINDOW, lParam));
           }
          else
           {
            return(DefWindowProc (hWnd, wMsg, wParam, lParam));
           }
          break;
         default:return(DefWindowProc (hWnd, wMsg, wParam, lParam));
        }
      case WM_COMMAND:
       switch(wParam)
        {
         case MC_EXIT:
```

```
         destroy_current_childs();
         DestroyWindow(hWnd);
         return(DefWindowProc (hWnd, wMsg, wParam, lParam));
        case MC_CLOCK:
         spawnlp(P_WAIT, "Clock.exe",(char *)NULL);
         return(DefWindowProc (hWnd, wMsg, wParam, lParam));
        case MC_NOTEPAD:
         spawnlp(P_WAIT, "Notepad.exe",(char *)NULL);
         return(DefWindowProc (hWnd, wMsg, wParam, lParam));
        case MC_CALCULATOR:
         spawnlp(P_WAIT, "Calc.exe",(char *)NULL);
         return(DefWindowProc (hWnd, wMsg, wParam, lParam));
        case MC_NEW_SESSION:
         destroy_current_childs();
         init_session();
         return(DefWindowProc (hWnd, wMsg, wParam, lParam));
        case MC_PREVIOUS_SESSION:
         destroy_current_childs();
         figflag = Ø;
         list_previous();
         return(DefWindowProc (hWnd, wMsg, wParam, lParam));
        case MC_MAP:
         lvm();
         return(DefWindowProc (hWnd, wMsg, wParam, lParam));
       }
     default:
          return(DefWindowProc (hWnd, wMsg, wParam, lParam));
          break;
   }
}

pascal WinMain (hInstance, hPrevInstance, lpCmdLine, nCmdShow)

  HANDLE hInstance, hPrevInstance;
  LPSTR lpCmdLine;
  int nCmdShow;

{
  WNDCLASS wndclass;
  MSG  msg;
  HMENU hmenu;
  HDC  hdc;
  LPSTR ptr;

  screen_height = GetSystemMetrics(SM_CYSCREEN);
  destroyed_flag = FALSE;
  no_of_open_files = Ø;
  current_active_window = Ø;
  current_f_coa = get_current_coa(fcoa, f_curvec);
  current_e_coa = get_current_coa(ecoa, e_curvec);
  current_highlight = NULL;
  next_session[Ø] = NULL;
  current_session[Ø] = NULL;
  if (hPrevInstance)
     {
       return(FALSE);
```

```
    }
  strcpy(id_name, "session");
  intacval_instance = hInstance;
  wndclass.style = Ø;
  wndclass.lpfnWndProc = HandleMainWindow;
  wndclass.cbClsExtra = Ø;
  wndclass.cbWndExtra = Ø;
  wndclass.hInstance = intacval_instance;
  wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
  wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
  wndclass.lpszMenuName = (LPSTR)"Main";
  wndclass.lpszClassName = (LPSTR)"MAIN";
  RegisterClass((LPWNDCLASS)&wndclass);
  wndclass.style = Ø;
  wndclass.lpfnWndProc = HandleFileWindow;
  wndclass.cbClsExtra = Ø;
  wndclass.cbWndExtra = 12;
  wndclass.hInstance = intacval_instance;
  wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
  wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
  wndclass.lpszMenuName = (LPSTR)NULL;
  wndclass.lpszClassName = (LPSTR)"FILES";
  RegisterClass((LPWNDCLASS)&wndclass);
  wndclass.style = Ø;
  wndclass.lpfnWndProc = HandleBoxesWindow;
  wndclass.cbClsExtra = Ø;
  wndclass.cbWndExtra = 12;
  wndclass.hInstance = intacval_instance;
  wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
  wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
  wndclass.lpszMenuName = (LPSTR)NULL;
  wndclass.lpszClassName = (LPSTR)"BOXES";
  RegisterClass((LPWNDCLASS)&wndclass);
  main_window = CreateWindow((LPSTR)"MAIN", (LPSTR)"INTACVAL",
                             WS_TILED|WS_CLIPCHILDREN,
                             Ø, Ø, Ø, Ø, NULL, NULL, intacval_instance,
                             (LPSTR)NULL);
  hdc = GetDC(main_window);
  SelectObject(hdc, GetStockObject(OEM_FIXED_FONT));
  GetTextMetrics(hdc, (LPTEXTMETRIC)&font_sizing);
  ReleaseDC(main_window, hdc);
  ShowWindow(main_window, nCmdShow);
  current_dialog_func = MakeProcInstance((FARPROC)main_dialog, intacval_instance
);
  current_figure = CreateDialog(intacval_instance, (LPSTR)"Figure2",
                  main_window,current_dialog_func);
  lvminit("vcorps.db");
   do {
     GetMessage((LPMSG)&msg, NULL, Ø, Ø);
     if (!IsDialogMessage(current_figure, (LPMSG)&msg))
      {
       if (!TranslateMessage((LPMSG)&msg))
          {
            DispatchMessage((LPMSG)&msg);
        }
    }
  } while ((msg.message != WM_QUIT) && IsWindow(main_window));
  FreeProcInstance(current_dialog_func);
  save_cur_session();
  return(msg.wParam);
}
```

```
#include "intacval.h"
#include "lvm.h"

pascal main_dialog(HWND, WORD, WORD, LONG);


delete_file(vec)
 char *vec;

 {
  int i,n;
  char filename[15];

  for (i = 0; i < 15; i++)
   {
    if (vec[i] == ' ') break;
    filename[i] = vec[i];
   }
  filename[i] = NULL;
  unlink(filename);
  SendMessage(main_window,WM_COMMAND,MC_PREVIOUS_SESSION,(LONG)(LPSTR)NULL);
 }

list_previous()
 {
  int i;
  HWND child_window;

  DestroyWindow(current_figure);
  FreeProcInstance(current_dialog_func);
  current_dialog_func = MakeProcInstance((FARPROC)main_dialog, intacval_instance
);
  get_dir(filenames,&i);
  current_figure = CreateDialog(intacval_instance, (LPSTR)"Figure3",
                                main_window,current_dialog_func);
  for(i = 0; i< 10; i++)
   {
    if (filenames[i][0] == ' ')
     break;
    SendDlgItemMessage(current_figure, DLG_FILELIST, LB_ADDSTRING,NULL,
      (LONG)(LPSTR)filenames[i]);
   }
 }

DIR_ENTRY *decipher (wildcard)

  char *wildcard;

 {
  static char last_wild[60] = "";
  union REGS srv, rrv;
  struct SREGS segv;
  static DIR_ENTRY dir_entry;
  char far *lpstr;

  lpstr = (char far *)&dir_entry;
```

```
    srv.x.ax = 0x1a00;
    srv.x.dx = FP_OFF(lpstr);
    segv.ds = FP_SEG(lpstr);
    intdosx(&srv, &rrv, &segv);
    srv.x.ax = 0x4e00;
    if (strcmp(wildcard, last_wild) == 0)
       {
         srv.x.ax = 0x4f00;
       }
      else
       {
         strcpy(last_wild,wildcard);
       }
    lpstr = (char far *)wildcard;
    srv.x.cx = 0;
    srv.x.dx = FP_OFF(lpstr);
    segv.ds = FP_SEG(lpstr);
    intdosx(&srv, &rrv, &segv);
    if (rrv.x.cflag)
       {
         return((DIR_ENTRY *)0);
       }
      else
       {
         return(&dir_entry);
       }
}

get_dir(filenames, pj)
 char filenames[][60];
 int *pj;

{
 int i, flag;
 char ext[3], filename[60];
 DIR_ENTRY *dir_entry;

 flag = FALSE;
 *pj = 0;
 for(i = 0; i < 10; i++)
  {
   initvec(filenames[i], 60);
  }
 for (i = 0; i < 100; i++)
  {
   inttostr(i,ext);
   ext[2] = NULL;
   strcpy(filename, id_name);
   strcat(filename,".");
   strcat(filename,ext);
 /*MessageBox(GetFocus(),(LPSTR)s,(LPSTR)"DEBUG",MB_OK);*/
   if (dir_entry = decipher(filename))
    {
     flag = FALSE;
     sprintf(filenames[*pj],"%-12s     %2d/%02d/%02d       %2d:%02d:%02d",
         dir_entry->name, dir_entry->date.day, dir_entry->date.month,
          dir_entry->date.year+8Ø, dir_entry->time.hour,
          dir_entry->time.minute, dir_entry->time.second*2);
      *pj += 1;
      if ((*pj) >=1Ø) return;
     }
    else
     {
      if (flag == FALSE)
       strcpy(next_session, filename);
      flag = TRUE;
     }
   }
}

inttostr(n,vec)
 int n;
 char *vec;

{
 vec[Ø] = n/1Ø + 'Ø';
 vec[1] = n%1Ø + 'Ø';
}


initvec(vec, n)
 char *vec;
 int n;

{
 int i;

 for(i = Ø; i < (n - 1); i++)
  {
   vec[i] = ' ';
  }
 vec[n-1] = NULL;
}


long pascal HandleFileWindow (hWnd, wMsg, wParam, lParam)

  HWND hWnd;
  WORD wMsg, wParam;
  LONG lParam;


{
  LONG retval;
  RECT window_rect;
  int  new_horz_max, new_vert_max;
  int fp, num_lines, num_cols;
  int new_i;

  switch (wMsg)
   {
     case WM_SYSCOMMAND:
      switch (wParam & Øxfff Ø)
       {
        case SC_KEYMENU:
         if (lParam == 9)
          {
           return(DefWindowProc (hWnd, wMsg, SC_NEXTWINDOW, lParam));
          }
         else
          {
           return(DefWindowProc (hWnd, wMsg, wParam, lParam));
          }
         break;
        case SC_MOVE:
         if (figflag < 13Ø)
            {
              switch_active_window(hWnd);
              return TRUE;
            }
           else
            {
              return(DefWindowProc(hWnd, wMsg, wParam, lParam));
            }
        case SC_CLOSE:
         delete_one_child(hWnd,&new_i);
         destroyed_flag = TRUE;
         if (figflag >= 13Ø)
          change_window_pos(hWnd,new_i);
         return(DefWindowProc (hWnd, wMsg, wParam, lParam));
        default:return(DefWindowProc (hWnd, wMsg, wParam, lParam));
       }
     case WM_PAINT:
          repaint(hWnd, (LPPAINTSTRUCT)lParam);
          return(1L);
          break;
     case WM_HSCROLL:
          side_scroll(hWnd, wParam, LOWORD(lParam));
          return(1L);
          break;
     case WM_VSCROLL:
          vert_scroll(hWnd, wParam, LOWORD(lParam));
          return(1L);
          break;
     case WM_SIZE:
          retval = DefWindowProc(hWnd, wMsg, wParam, lParam);
          switch (wParam)
           {
             case SIZEFULLSCREEN:
             case SIZENORMAL:
                  new_vert_max = GetWindowWord(hWnd, 2);
                  new_horz_max = GetWindowWord(hWnd, 4);
                  new_vert_max = GetWindowWord(hWnd, 2);
                  new_horz_max -= LOWORD(lParam)/font_sizing.tmMaxCharWidth;
                  new_vert_max -= HIWORD(lParam)/font_sizing.tmHeight;
                  if (new_horz_max < Ø) new_horz_max = Ø;
                  if (new_vert_max < Ø) new_vert_max = Ø
                  if ((GetScrollPos(hWnd, SB_HORZ) > new_horz_max)
                        ||(GetScrollPos(hWnd, SB_VERT) > new_vert_max))
                         {
                           SetScrollPos(hWnd, SB_HORZ, new_horz_max, FALSE);
                           SetScrollPos(hWnd, SB_VERT, new_vert_max, FALSE);
                           GetClientRect(hWnd, (LPRECT)&window_rect);
                           InvalidateRect(hWnd, (LPRECT)&window_rect, FALSE);
                         }
                     SetScrollRange(hWnd, SB_HORZ, 0, new_horz_max, TRUE);
                     SetScrollRange(hWnd, SB_VERT, 0, new_vert_max, TRUE);
                     break;
                default:
                     break;
              }
            return(retval);
            break;
        case WM_DESTROY:
            fp = GetWindowWord(hWnd, 0);
            close(fp);
            if (destroyed_flag == FALSE)
                delete_one_child(hWnd,&new_i);
            destroyed_flag = FALSE;
            return(DefWindowProc (hWnd, wMsg, wParam, lParam));
        case WM_CREATE:
            SetWindowWord(hWnd, 0, new_fp);
            scanfile(new_fp, &num_lines, &num_cols);
            SetWindowWord(hWnd, 2, num_lines);
            SetWindowWord(hWnd, 4, num_cols);
            SetWindowWord(hWnd, 6, 0);
            SetWindowLong(hWnd, 8, 0l);
            BringWindowToTop(hWnd);
            return(DefWindowProc (hWnd, wMsg, wParam, lParam));
        case WM_LBUTTONDOWN:
            if (figflag < 130)
              switch_active_window(hWnd);
            return TRUE;
        default:
            return(DefWindowProc (hWnd, wMsg, wParam, lParam));
            break;
    }
}

transbuffer(buffer, longbuffer)
  char *buffer, *longbuffer;

{
  int i, j;

  i = 0;
  j = 0;
  while((buffer[i] != '\r') && (buffer[i] != 0x1a) && (buffer[i] != NULL))
   {

     if (buffer[i] == '\t')
      {
        j &= ~7;
        j += 8;
        i++;
      }
     else
      {
        longbuffer[j++] = buffer[i++];
      }
   }
}

two_windows_one_button(button)
int button;

{
 char str[3Ø];

 switch(figflag)
  {
   case 13Ø:
    {
     switch (button)
      {
       case DLG_BUTTON1:
        {
         strcpy(str, "DISPLAY KB'S");
         strcpy(s,"Click on 'DISPLAY KB'S' to review KB's ");
         strcat(s," or to change values.");
         get_f_coa(1);
         figflag = 135;
         break;
        }
       case DLG_BUTTON2:
        {
         strcpy(str, "SEE BLUE COA");
         strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
         strcat(s, " Blue COA.");
         display_oav(1);
         figflag = 136;
         break;
        }
       default:
        break;
      }
     break;
    }
   case 131:
    {
     switch (button)
      {
       case DLG_BUTTON1:
        {
         strcpy(str, "SEE BLUE COA");
         strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
         strcat(s, " Blue COA.");
         file_handle[1] = file_handle[Ø];
         get_e_coa(Ø);
         figflag = 136;
         break;
        }
       case DLG_BUTTON2:
        {
         strcpy(str, "SEE RED COA");
         strcpy(s,"Click on 'SEE RED COA' to view corresponding");
         strcat(s, " Red COA.");
         file_handle[1] = file_handle[Ø];
         get_f_coa(Ø);
         figflag = 137;
         break;
        }
       default:
        break;
      }
     break;
    }
   case 132:
    {
     switch (button)
      {
       case DLG_BUTTON1:
        {
         strcpy(str, "DISPLAY KB'S");
         strcpy(s,"Click on 'DISPLAY KB'S' to review KB's ");
         strcat(s," or to change values.");
         file_handle[1] = file_handle[Ø];
         get_e_coa(Ø);
         figflag = 135;
         break;
        }
       case DLG_BUTTON2:
        {
         strcpy(str, "SEE RED COA");
         strcpy(s,"Click on 'SEE RED COA' to view corresponding");
         strcat(s, " Red COA.");
         display_oav(1);
         figflag = 137;
         break;
        }
       default:
        break;
      }
     break;
    }
   case 138:
    {
     switch(button)
      {
       case Ø:
        {
         sprintf(str,"%s%d",FRIENDLY_FILE_NAME,current_f_coa);
         change_file(file_handle[Ø], str);
         show_mask &= ~Øx2ØØØ;
         redo_screen();
         strcpy(str, "SEE RED COA");
```

```
          strcpy(s,"Click on 'SEE RED COA' to view corresponding");
          strcat(s, " Red COA.");
          figflag = 137;
          break;
         }
        case 1:
         {
          sprintf(str,"%s%d",ENEMY_FILE_NAME,current_e_coa);
          change_file(file_handle[Ø], str);
          show_mask &= ~Øx ØØ2Ø;
          redo_screen();
          strcpy(str, "SEE BLUE COA");
          strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
          strcat(s, " Blue COA.");
          figflag = 136;
          break;
         }
        case 2:
         {
          sprintf(str,"%s%d",ENEMY_FILE_NAME,current_e_coa);
          change_file(file_handle[Ø], str);
          sprintf(str,"%s%d",FRIENDLY_FILE_NAME,current_f_coa);
          change_file(file_handle[1], str);
          strcpy(str, "DISPLAY KB'S");
          strcpy(s,"Click on 'DISPLAY KB'S' to review KB's ");
          strcat(s," or to change values.");
          figflag = 135;
          break;
         }
        default:
         break;
       }
      break;
     }
    case 14Ø:
     {
      switch (button)
       {
        case DLG_BUTTON1:
         {
          strcpy(str, "DISPLAY KB'S");
          strcpy(s,"Click on 'DISPLAY KB'S' to review KB's ");
          strcat(s," or to change values.");
          get_e_coa(1);
          figflag = 145;
          break;
         }
        case DLG_BUTTON2:
         {
          strcpy(str, "SEE RED COA");
          strcpy(s,"Click on 'SEE RED COA' to view corresponding");
          strcat(s, " Red COA.");
          display_oav(1);
          figflag = 146;
          break;
         }
        default:
         break;
       }
      break;
     }
    case 141:
     {
      switch (button)
       {
        case DLG_BUTTON1:
         {
          strcpy(str, "SEE RED COA");
          strcpy(s,"Click on 'SEE RED COA' to view corresponding");
          strcat(s, " Red COA.");
          file_handle[1] = file_handle[Ø];
          get_f_coa(Ø);
          figflag = 146;
          break;
         }
        case DLG_BUTTON2:
         {
          strcpy(str, "SEE BLUE COA");
          strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
          strcat(s, " Blue COA.");
          file_handle[1] = file_handle[Ø];
          get_e_coa(Ø);
          figflag = 147;
          break;
         }
        default:
         break;
       }
      break;
     }
    case 142:
     {
      switch (button)
       {
        case DLG_BUTTON1:
         {
          strcpy(str, "DISPLAY KB'S");
          strcpy(s,"Click on 'DISPLAY KB'S' to review KB's ");
          strcat(s," or to change values.");
          file_handle[1] = file_handle[Ø];
          get_f_coa(Ø);
          figflag = 145;
          break;
         }
        case DLG_BUTTON2:
         {
          strcpy(str, "SEE BLUE COA");
          strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
          strcat(s, " Blue COA.");
          display_oav(1);
          figflag = 147;
          break;
        }
       default:
        break;
      }
     break;
    }
   case 148:
    {
     switch(button)
      {
       case 0:
        {
         sprintf(str,"%s%d",ENEMY_FILE_NAME,current_e_coa);
         change_file(file_handle[0], str);
         show_mask &= ~0x0020;
         redo_screen();
         strcpy(str, "SEE BLUE COA");
         strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
         strcat(s, " Blue COA.");
         figflag = 147;
         break;
        }
       case 1:
        {
         sprintf(str,"%s%d",FRIENDLY_FILE_NAME,current_f_coa);
         change_file(file_handle[0], str);
         show_mask &= ~0x2000;
         redo_screen();
         strcpy(str, "SEE RED COA");
         strcpy(s,"Click on 'SEE RED COA' to view corresponding");
         strcat(s, " Red COA.");
         figflag = 146;
         break;
        }
       case 2:
        {
         sprintf(str,"%s%d",FRIENDLY_FILE_NAME,current_f_coa);
         change_file(file_handle[0], str);
         sprintf(str,"%s%d",ENEMY_FILE_NAME,current_e_coa);
         change_file(file_handle[1], str);
         strcpy(str, "DISPLAY KB'S");
         strcpy(s,"Click on 'DISPLAY KB'S' to review KB's ");
         strcat(s," or to change values.");
         figflag = 145;
         break;
        }
       default:
        break;
      }
     break;
    }
  }
  DestroyWindow(current_figure);
  FreeProcInstance(current_dialog_func);
  current_dialog_func =
     MakeProcInstance((FARPROC)main_dialog, intacval_instance);
  current_figure = CreateDialog(intacval_instance,
    (LPSTR)"Figure14", main_window,current_dialog_func);
  SetDlgItemText(current_figure, DLG_BUTTON, (LPSTR)str);
  SetDlgItemText(current_figure, DLG_CLICK, (LPSTR)s);
  MoveWindow(file_handle[Ø],94,-1,542,65*screen_height/2ØØ,TRUE);
  MoveWindow(file_handle[1],94,65*screen_height/2ØØ-2,542,65*screen_height/2ØØ,T
RUE);
  SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
       TRUE,(LONG)NULL);
}

find_which_window(hwnd,pi)
HWND hwnd;
int *pi;

{
 int i;

  for(i = Ø; i < no_of_open_files; i++)
   {
    if (file_handle[i] == hwnd)
     {
      *pi = i;
      break;
     }
   }
}


one_window_two_buttons(button)
int button;

{
 char str1[3Ø], str2[3Ø];

 switch(button)
  {
   case DLG_E_COA:
        {
         strcpy(str1, "SEE BLUE COA");
         strcpy(str2, "DISPLAY KB'S");
         strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
         strcat(s, " Blue COA.                    Click on 'DISPLAY ");
         strcat(s, "KB'S' to  review KB's or to change values.");
         get_e_coa(Ø);
         figflag = 13Ø;
         break;
        }
   case DLG_F_COA:
        {
         strcpy(str1, "SEE RED COA");
         strcpy(str2, "DISPLAY KB'S");
         strcpy(s,"Click on 'SEE RED COA' to view corresponding");
         strcat(s, " Red COA.                     Click on 'DISPLAY ");
         strcat(s, "KB'S' to  review KB's or to change values.");
         get_f_coa(Ø);
         figflag = 14Ø;
         break;
        }
    case Ø:
     {
      switch(figflag)
       {
        case 135:
         {
          show_mask &= ~Øx2ØØØ;
          redo_screen();
          strcpy(str1, "SEE RED COA");
          strcpy(str2, "DISPLAY KB'S");
          strcpy(s,"Click on 'SEE RED COA' to view corresponding");
          strcat(s, " Red COA.                         Click on 'DISPLAY ");
          strcat(s, "KB'S' to  review KB's or to change values.");
          figflag = 132;
          break;
         }
       case 136: case 137:
         {
          show_mask &= ~Øx2Ø2Ø;
          redo_screen();
          strcpy(str1, "SEE RED COA");
          strcpy(str2, "SEE BLUE COA");
          strcpy(s,"Click on 'SEE RED COA' to view corresponding");
          strcat(s, " Red COA.                         Click on 'SEE ");
          strcat(s, "BLUE COA' to view corresponding Blue COA");
          figflag = 131;
          break;
         }
       case 145:
         {
          show_mask &= ~ØxØØ2Ø;
          redo_screen();
          strcpy(str1, "SEE BLUE COA");
          strcpy(str2, "DISPLAY KB'S");
          strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
          strcat(s, " Blue COA.                        Click on 'DISPLAY ");
          strcat(s, "KB'S' to  review KB's or to change values.");
          figflag = 142;
          break;
         }
       case 146: case 147:
         {
          show_mask &= ~Øx2Ø2Ø;
          redo_screen();
          strcpy(str1, "SEE BLUE COA");
          strcpy(str2, "SEE RED COA");
          strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
          strcat(s, " Blue COA.                        Click on 'SEE ");
          strcat(s, "RED COA' to view corresponding Red COA");
          figflag = 141;
          break;
         }
       }
      break;
     }
    case 1:
     {
      switch(figflag)
       {
        case 135: case 136:
         {
          show_mask &= ~Øx ØØ2Ø;
          redo_screen();
          strcpy(str1, "SEE BLUE COA");
          strcpy(str2, "DISPLAY KB'S");
          strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
          strcat(s, " Blue COA.                              Click on 'DISPLAY ");
          strcat(s, "KB'S' to  review KB's or to change values.");
          figflag = 13Ø;
          break;
         }
       case 137:
         {
          show_mask &= ~Øx2ØØØ;
          redo_screen();
          strcpy(str1, "SEE RED COA");
          strcpy(str2, "DISPLAY KB'S");
          strcpy(s,"Click on 'SEE RED COA' to view corresponding");
          strcat(s, " Red COA.                                Click on 'DISPLAY ");
          strcat(s, "KB'S' to  review KB's or to change values.");
          figflag = 132;
          break;
         }
       case 145: case 146:
         {
          show_mask &= ~Øx2ØØØ;
          redo_screen();
          strcpy(str1, "SEE RED COA");
          strcpy(str2, "DISPLAY KB'S");
          strcpy(s,"Click on 'SEE RED COA' to view corresponding");
          strcat(s, " Red COA.                                Click on 'DISPLAY ");
          strcat(s, "KB'S' to  review KB's or to change values.");
          figflag = 14Ø;
          break;
         }
       case 147:
         {
          show_mask &= ~ØxØØ2Ø;
          redo_screen();
          strcpy(str1, "SEE BLUE COA");
          strcpy(str2, "DISPLAY KB'S");
          strcpy(s,"Click on 'SEE BLUE COA' to view corresponding");
          strcat(s, " Blue COA.                              Click on 'DISPLAY ");
          strcat(s, "KB'S' to  review KB's or to change values.");
          figflag = 142;
          break;
         }
       }
      break;
     }
   }
  DestroyWindow(current_figure);
  FreeProcInstance(current_dialog_func);
  current_dialog_func =
     MakeProcInstance((FARPROC)main_dialog, intacval_instance);
  current_figure = CreateDialog(intacval_instance,
    (LPSTR)"Figure13", main_window,current_dialog_func);
  SetDlgItemText(current_figure, DLG_BUTTON1, (LPSTR)str1);
  SetDlgItemText(current_figure, DLG_BUTTON2, (LPSTR)str2);
  SetDlgItemText(current_figure, DLG_CLICK, (LPSTR)s);
  MoveWindow(file_handle[0],94,-1,542,130*screen_height/200,TRUE);
  SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
        TRUE,(LONG)NULL);
}

three_windows_no_buttons()

{
 char str[15];

 switch(figflag)
  {
   case 135:
    {
     sprintf(str,"%s%d",SENEMY_FILE_NAME,current_e_coa);
     change_file(file_handle[0],str);
     sprintf(str,"%s%d",SFRIENDLY_FILE_NAME,current_f_coa);
     change_file(file_handle[1],str);
     display_oav(2);
     figflag = 138;
     break;
    }
   case 136:
    {
     sprintf(str,"%s%d",SENEMY_FILE_NAME,current_e_coa);
     change_file(file_handle[0], str);
     file_handle[2] = file_handle[1];
     get_sf_coa(1);
     figflag = 138;
     break;
    }
   case 137:
    {
     file_handle[2] = file_handle[1];
     file_handle[1] = file_handle[0];
     sprintf(str,"%s%d",SFRIENDLY_FILE_NAME,current_f_coa);
     change_file(file_handle[1], str);
     get_se_coa(0);
     figflag = 138;
     break;
    }
   case 145:
    {
     sprintf(str,"%s%d",SENEMY_FILE_NAME,current_e_coa);
     change_file(file_handle[1], str);
```

```
        sprintf(str,"%s%d",SFRIENDLY_FILE_NAME,current_f_coa);
        change_file(file_handle[Ø], str);
        display_oav(2);
        figflag = 148;
        break;
       }
     case 146:
       {
        file_handle[2] = file_handle[1];
        sprintf(str,"%s%d",SFRIENDLY_FILE_NAME,current_f_coa);
        change_file(file_handle[Ø], str);
        get_se_coa(1);
        figflag = 148;
        break;
       }
     case 147:
       {
        file_handle[2] = file_handle[1];
        file_handle[1] = file_handle[Ø];
        sprintf(str,"%s%d",SENEMY_FILE_NAME,current_e_coa);
        change_file(file_handle[1], str);
        get_sf_coa(Ø);
        figflag = 148;
        break;
       }
    }
  DestroyWindow(current_figure);
  FreeProcInstance(current_dialog_func);
  current_dialog_func =
     MakeProcInstance((FARPROC)main_dialog, intacval_instance);
  current_figure = CreateDialog(intacval_instance,
    (LPSTR)"Figure5", main_window,current_dialog_func);
  strcpy(s, "Objects and attributes remain fixed. Click on ");
  strcat(s, "desired value box to select new value. Use horizontal ");
  strcat(s, "scroll bar to view more KB's.");
  SetDlgItemText(current_figure, DLG_CLICK, (LPSTR)s);
  MoveWindow(file_handle[Ø],94,-1,272,67*screen_height/2ØØ,TRUE);
  MoveWindow(file_handle[1],364,-1,272,67*screen_height/2ØØ,TRUE);
  MoveWindow(file_handle[2],94,67*screen_height/2ØØ-2,542,82*screen_height/2ØØ,TR
UE);
  SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
        TRUE,(LONG)NULL);
 }
```

```
#include "intacval.h"
#include "lvm.h"

get_mission_files()

 {

  char file_caption[3][2Ø];
  HANDLE ltext_handle;
  int fp;

  destroy_current_childs();
  checkfigure();
  if (current_highlight != NULL)
    SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
        FALSE,(LONG)NULL);
  current_highlight = DLG_MISSION;
  SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
        TRUE,(LONG)NULL);
  no_of_open_files = 3;
  current_active_window = 1;
  strcpy(s, "To review text, click on the tittle of the window containing ");
  strcat(s, "the text you want to see. Click on the arrows to scroll text.");
  SetDlgItemText(current_figure, DLG_CLICK, (LPSTR)s);
  new_fp = open(OBJ_FILE_NAME, O_BINARY);
  file_handle[2] = CreateWindow((LPSTR)"FILES", (LPSTR)"OBJECTIVES",
        WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|WS_VISIBLE,
        14Ø, 5*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
        intacval_instance,(LPSTR)NULL);
  new_fp = open(INT_FILE_NAME, O_BINARY);
  file_handle[1] = CreateWindow((LPSTR)"FILES", (LPSTR)"INTENT",
        WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|WS_VISIBLE,
        13Ø, 13*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
        intacval_instance,(LPSTR)NULL);
  new_fp = open(ASS_FILE_NAME, O_BINARY);
  file_handle[Ø] = CreateWindow((LPSTR)"FILES", (LPSTR)"ASSUMPTIONS",
        WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|WS_VISIBLE,
        12Ø, 21*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
        intacval_instance,(LPSTR)NULL);
 }

get_terrain_files()

 {

  char file_caption[3][2Ø];
  HANDLE ltext_handle;
  int fp;

  destroy_current_childs();
  checkfigure();
  if (current_highlight != NULL)
    SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
        FALSE,(LONG)NULL);
  show_mask = (1 << TERRAIN);
  redo_screen();
```

```
    current_highlight = DLG_TERRAIN;
    SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
         TRUE,(LONG)NULL);
    no_of_open_files = 2;
    current_active_window = 1;
    strcpy(s, "To review text, click on the tittle of the window containing ");
    strcat(s, "the text you want to see. Click on the arrows to scroll text.");
    SetDlgItemText(current_figure, DLG_CLICK, (LPSTR)s);
    new_fp = open(GEN_FILE_NAME, O_BINARY);
    file_handle[1] = CreateWindow((LPSTR)"FILES", (LPSTR)"KEY TERRAIN",
         WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|WS_VISIBLE,
         14Ø, 5*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
         intacval_instance,(LPSTR)NULL);
    new_fp = open(KEY_FILE_NAME, O_BINARY);
    file_handle[Ø] = CreateWindow((LPSTR)"FILES", (LPSTR)"GENERAL TERRAIN",
         WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|WS_VISIBLE,
         13Ø, 13*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
         intacval_instance,(LPSTR)NULL);
  }

get_friendly_files()

  {

    char file_caption[3][2Ø];
    HANDLE ltext_handle;
    int fp;

    destroy_current_childs();
    checkfigure();
    if (current_highlight != NULL)
      SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
         FALSE,(LONG)NULL);
    show_mask = Øx1f;
    redo_screen();
    current_highlight = DLG_F_CAP;
    SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
         TRUE,(LONG)NULL);
    no_of_open_files = 3;
    current_active_window = 1;
    strcpy(s, "To review text, click on the tittle of the window containing ");
    strcat(s, "the text you want to see. Click on the arrows to scroll text.");
    SetDlgItemText(current_figure, DLG_CLICK, (LPSTR)s);
    new_fp = open(FCON_FILE_NAME, O_BINARY);
    file_handle[2] = CreateWindow((LPSTR)"FILES", (LPSTR)"CONDITION",
         WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|WS_VISIBLE,
         14Ø, 5*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
         intacval_instance,(LPSTR)NULL);
    new_fp = open(FREI_FILE_NAME, O_BINARY);
    file_handle[1] = CreateWindow((LPSTR)"FILES", (LPSTR)"REINFORCEMENTS",
         WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|WS_VISIBLE,
         13Ø, 13*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
         intacval_instance,(LPSTR)NULL);
    new_fp = open(FCOM_FILE_NAME, O_BINARY);
    file_handle[Ø] = CreateWindow((LPSTR)"FILES",
         (LPSTR)"COMPOSITION/LOCATION/DISPOSITION",
        WS_CHILD!WS_SYSMENU!WS_CLIPSIBLINGS!WS_CAPTION!WS_VISIBLE,
        12Ø, 21*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
        intacval_instance,(LPSTR)NULL);
 }


get_enemy_files()

 {

  char file_caption[3][2Ø];
  HANDLE ltext_handle;
  int fp;

  destroy_current_childs();
  checkfigure();
  if (current_highlight != NULL)
    SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
       FALSE,(LONG)NULL);
  show_mask = Øx1fØØ;
  redo_screen();
  current_highlight = DLG_E_CAP;
  SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
       TRUE,(LONG)NULL);
  no_of_open_files = 3;
  current_active_window = 1;
  strcpy(s, "To review text, click on the tittle of the window containing ");
  strcat(s, "the text you want to see. Click on the arrows to scroll text.");
  SetDlgItemText(current_figure, DLG_CLICK, (LPSTR)s);
  new_fp = open(ECON_FILE_NAME, O_BINARY);
  file_handle[2] = CreateWindow((LPSTR)"FILES", (LPSTR)"CONDITION",
       WS_CHILD!WS_SYSMENU!WS_CLIPSIBLINGS!WS_CAPTION!WS_VISIBLE,
       14Ø, 5*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
       intacval_instance,(LPSTR)NULL);
  new_fp = open(EREI_FILE_NAME, O_BINARY);
  file_handle[1] = CreateWindow((LPSTR)"FILES", (LPSTR)"REINFORCEMENTS",
       WS_CHILD!WS_SYSMENU!WS_CLIPSIBLINGS!WS_CAPTION!WS_VISIBLE,
       13Ø, 13*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
       intacval_instance,(LPSTR)NULL);
  new_fp = open(ECOM_FILE_NAME, O_BINARY);
  file_handle[Ø] = CreateWindow((LPSTR)"FILES",
       (LPSTR)"COMPOSITION/LOCATION/DISPOSITION",
       WS_CHILD!WS_SYSMENU!WS_CLIPSIBLINGS!WS_CAPTION!WS_VISIBLE,
       12Ø, 21*screen_height/2ØØ, 48Ø, 12Ø*screen_height/2ØØ, main_window, NULL,
       intacval_instance,(LPSTR)NULL);
  }

 get_e_coa(n)
  int n;

  {
   char str1[3Ø], str2[3Ø];

   show_mask &= Øx2Ø2Ø;
   show_mask != Øx3f1f;
   redo_screen();
  no_of_open_files++;
  strcpy(str1,ENEMY_FILE_NAME);
  sprintf(str2,"%s%d",str1,current_e_coa);
  new_fp = open(str2, O_RDONLY | O_BINARY);
  sprintf(str1, "RED COA #%d",current_e_coa);
  file_handle[n] = CreateWindow((LPSTR)"FILES", (LPSTR)str1,
                   WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|
                   WS_VISIBLE, 1ØØ, Ø,5,5, main_window,
                   NULL, intacval_instance,(LPSTR)NULL);
 }

get_se_coa(n)
 int n;

 {
  char str1[3Ø], str2[3Ø];

  show_mask &= Øx2Ø2Ø;
  show_mask |= Øx3f1f;
  redo_screen();
  no_of_open_files++;
  strcpy(str1,SENEMY_FILE_NAME);
  sprintf(str2,"%s%d",str1,current_e_coa);
  new_fp = open(str2, O_RDONLY | O_BINARY);
  sprintf(str1, "RED COA #%d",current_e_coa);
  file_handle[n] = CreateWindow((LPSTR)"FILES", (LPSTR)str1,
                   WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|
                   WS_VISIBLE, 1ØØ, Ø, 5, 5, main_window,
                   NULL, intacval_instance,(LPSTR)NULL);
 }

get_f_coa(n)
 int n;

 {
  char str1[3Ø], str2[3Ø];

  show_mask &= Øx2Ø2Ø;
  show_mask |= Øx1f3f;
  redo_screen();
  no_of_open_files++;
  strcpy(str1,FRIENDLY_FILE_NAME);
  sprintf(str2,"%s%d",str1,current_f_coa);
  new_fp = open(str2, O_RDONLY | O_BINARY);
  sprintf(str1, "BLUE COA #%d",current_f_coa);
  file_handle[n] = CreateWindow((LPSTR)"FILES", (LPSTR)str1,
                   WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|
                   WS_VISIBLE, 1ØØ, Ø, 5, 5, main_window,
                   NULL, intacval_instance,(LPSTR)NULL);
 }

get_sf_coa(n)
 int n;

 {
  char str1[3Ø], str2[3Ø];
  show_mask &= Øx2Ø2Ø;
  show_mask |= Øx1f3f;
  redo_screen();
  no_of_open_files++;
  strcpy(str1,SFRIENDLY_FILE_NAME);
  sprintf(str2,"%s%d",str1,current_f_coa);
  new_fp = open(str2, O_RDONLY | O_BINARY);
  sprintf(str1, "BLUE COA #%d",current_f_coa);
  file_handle[n] = CreateWindow((LPSTR)"FILES", (LPSTR)str1,
                   WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|
                   WS_VISIBLE, 1ØØ, Ø, 5, 5, main_window,
                   NULL, intacval_instance,(LPSTR)NULL);
 }

switch_active_window(hwnd)

 HWND hwnd;

 {
  int i, new_i[3];
  int fp[3], num_lines[3], current_line[3];
  long filepointer[3];
  char text[3][5Ø];
  int maxpos[3], minpos[3], pos[3];

  for(i = Ø; i < no_of_open_files; i++)
   {
    if (file_handle[i] == hwnd)
     {
      new_i[Ø] = i;
      break;
     }
   }
  if (i == Ø) return;
  switch_current(i);
  if (no_of_open_files == 2)
   switch2(i, new_i);
  else
   switch3(i, new_i);
  for(i = Ø; i < no_of_open_files; i++)
   {
    fp[i] = GetWindowWord(file_handle[new_i[i]], Ø);
    num_lines[i] = GetWindowWord(file_handle[new_i[i]], 2);
    current_line[i] =  GetWindowWord(file_handle[new_i[i]], 6);
    filepointer[i] = GetWindowLong(file_handle[new_i[i]], 8);
    GetWindowText(file_handle[new_i[i]],(LPSTR)text[i], 5Ø);
    GetScrollRange(file_handle[new_i[i]], SB_VERT, (LPINT)(minpos+i),
      (LPINT)(maxpos+i));
    pos[i] = GetScrollPos(file_handle[new_i[i]], SB_VERT);
   }
  for(i = Ø; i < no_of_open_files; i++)
   {
    SetWindowWord(file_handle[i], Ø, fp[i]);
    SetWindowWord(file_handle[i], 2, num_lines[i]);
    SetWindowWord(file_handle[i], 6, current_line[i]);
```

```
     SetWindowLong(file_handle[i], 8, filepointer[i]);
     SetWindowText(file_handle[i], (LPSTR)text[i]);
     SetScrollRange(file_handle[i], SB_VERT, minpos[i],maxpos[i], FALSE);
     SetScrollPos(file_handle[i], SB_VERT, pos[i], TRUE);
     InvalidateRect(file_handle[i], (LPRECT)NULL, NULL);
    }
  }

switch2(i, new_i)
 int i, *new_i;

 {
  new_i[Ø] = 1;
  new_i[1] = Ø;
 }

switch3(i, new_i)
 int i, *new_i;

 {
  if (i == 1)
   {
    new_i[Ø] = 1;
    new_i[1] = 2;
    new_i[2] = Ø;
   }
  else
   {
    new_i[Ø] = 2;
    new_i[1] = Ø;
    new_i[2] = 1;
   }
 }

switch_current(i)
 int i;

 {
  int n;
  n = current_active_window + i;
  if (n > no_of_open_files)
   current_active_window = n % (no_of_open_files - 1);
  else
   current_active_window = n;
   return;
 }

destroy_current_childs()

 {
  int i, n;
  n = no_of_open_files;
  for(i = n-1; i >= Ø; i--)
   {
    DestroyWindow(file_handle[i]);
   }
  show_mask = Ø;
  redo_screen();
 }

delete_one_child(hwnd,pi)

  HWND hwnd;
  int *pi;

 {
  int i,new_i;

  find_which_window(hwnd, &new_i);
  *pi = new_i;
  if (new_i < (no_of_open_files - 1))
   {
     for (i = new_i; i < (no_of_open_files - 1); i++)
      {
       file_handle[i] = file_handle[i+1];
      }
   }
  no_of_open_files--;
  return,
 }

change_window_pos(hwnd,i)
 HWND hwnd;
 int i;

 {


   switch(figflag)
    {
     case 13Ø: case 131: case 132: case 14Ø: case 141: case 142:
      {
       PostMessage(current_figure,WM_COMMAND,DLG_START,(LONG)(LPSTR)NULL);
       break;
      }
     case 135: case 136: case 137: case 145: case 146: case 147:
      {
       one_window_two_buttons(i);
       break;
      }
     case 138: case 148:
      {
       two_windows_one_button(i);
       break;
      }
     default:
      break;
    }
  }

update_opposing_forces(new_curvec, old_curvec)
 unsigned char new_curvec[], old_curvec[];
```

```
{
 int i;

 for (i = 3; i < 8; i++)
  {
    old_curvec[i] = new_curvec[i+5];
  }
 for (i = 8; i < 13; i++)
  {
    old_curvec[i] = new_curvec[i-5];
  }
}

checkfigure()

 {
  if (figflag < 13Ø) return;
  SendMessage(current_figure, WM_COMMAND, DLG_START, (LONG)(LPSTR)NULL);
 }

continue_func()

 {
  if ((current_active_window < no_of_open_files)
   && (figflag < 13Ø))
   {
    switch_active_window(file_handle[1]);
    if (current_highlight)
       {
         SendDlgItemMessage(current_figure, current_highlight, BM_SETSTATE,
              TRUE,(LONG)NULL);
       }
    return;
   }
  if ((figflag < 5Ø) || (figflag > 148))
   {
    return;
   }
  switch(figflag)
   {
    case 5Ø:
      PostMessage(current_figure,WM_COMMAND,DLG_MISSION,(LONG)(LPSTR)NULL);
      return;
    case 6Ø:
      PostMessage(current_figure,WM_COMMAND,DLG_TERRAIN,(LONG)(LPSTR)NULL);
      return;
    case 9Ø:
      PostMessage(current_figure,WM_COMMAND,DLG_F_CAP,(LONG)(LPSTR)NULL);
      return;
    case 11Ø:
      PostMessage(current_figure,WM_COMMAND,DLG_E_CAP,(LONG)(LPSTR)NULL);
      return;
    case 12Ø:
      PostMessage(current_figure,WM_COMMAND,DLG_E_COA,(LONG)(LPSTR)NULL);
      return;
      case 13Ø: case 131: case 132: case 14Ø: case 141: case 142:
        PostMessage(current_figure,WM_COMMAND,DLG_BUTTON1,(LONG)(LPSTR)NULL);
        return;
      case 135: case 136: case 137: case 145: case 146: case 147:
        PostMessage(current_figure,WM_COMMAND,DLG_BUTTON,(LONG)(LPSTR)NULL);
        return;
      case 138:
        PostMessage(current_figure,WM_COMMAND,DLG_F_COA,(LONG)(LPSTR)NULL);
        return;
      default:
        return;
    }
  }
```

```
#include "intacval.h"

scanfile(fp, plines, pcols)

  int fp;
  int *plines, *pcols;

{
  char buffer[MAXLENGTH];
  int i, j, n, maxcols;

  maxcols = Ø;
  *plines = Ø;
  while((n = getline(buffer, MAXLENGTH, fp)) != EOF)
   {
     *plines += 1;
     j = Ø;
     for(i = Ø; i < n; i++)
      {
        if (buffer[i] != '\t')
           j++;
        else
         {
           j &= ~7;
           j += 8;
         }
      }
     if (j > maxcols) maxcols = j;
   }
  *pcols = maxcols;
}


getline(buffer, maxlength, fp)

  int fp, maxlength;
  char *buffer;

{
  int n;
  long cpos, newpos;
  char *p;
  n = read(fp, buffer, maxlength - 1);
  if (n <= Ø)
   {
    buffer[Ø] = NULL;
    return EOF;
   }
  buffer[n] = NULL;
  cpos = lseek(fp, Ø1, 1);
  p = strchr(buffer, '\n');
  if (p)
   {
     newpos = cpos - (long)(n - (p - buffer) - 2) - 1;
     lseek(fp, newpos, Ø);
     return(int)(p - buffer);
   }
  return n;
}

repaint (hwnd, lpArea)

  HWND hwnd;
  LPPAINTSTRUCT lpArea;

{
  char buffer[MAXLENGTH + 5];
  char longbuffer[MAXLENGTH + 5];
  int  i, j, start_col, num_cols, base_col, top_edge;
  int  start_line, num_lines, base_line, last_base_line;
  int fp, n;
  long basepos, lastbasepos;

  base_col = GetScrollPos(hwnd, SB_HORZ);
  base_line = GetScrollPos(hwnd, SB_VERT);
  BeginPaint(hwnd, lpArea);
  SelectObject(lpArea->hdc, GetStockObject(OEM_FIXED_FONT));
  start_col = lpArea->rcPaint.left/font_sizing.tmMaxCharWidth;
  num_cols = lpArea->rcPaint.right/font_sizing.tmMaxCharWidth-start_col+1;
  top_edge = (lpArea->rcPaint.top/font_sizing.tmHeight)*font_sizing.tmHeight;
  start_line = lpArea->rcPaint.top/font_sizing.tmHeight;
  if (start_line < Ø)
   {
    EndPaint(hwnd, lpArea);
    return;
   }
  num_lines = lpArea->rcPaint.bottom/font_sizing.tmHeight-start_line+1;
  fp = GetWindowWord(hwnd, Ø);
  set_new_base(hwnd, fp, base_line, &basepos);
  basepos = lseek(fp, Øl, 1);
  SetWindowLong(hwnd, 8, basepos);
  SetWindowWord(hwnd, 6, base_line);
  for (i = Ø; i < start_line; i++)
   {
     getline(buffer, MAXLENGTH, fp);
   }
  for (i = Ø; i < num_lines; i++)
   {
     n = getline(buffer, MAXLENGTH, fp);
     initvec(longbuffer, MAXLENGTH);
     transbuffer(buffer, longbuffer);
     TextOut(lpArea->hdc, start_col * font_sizing.tmMaxCharWidth,
             (i  + start_line)* font_sizing.tmHeight,
             (LPSTR)(longbuffer + base_col+start_col), num_cols);
   }
  EndPaint(hwnd, lpArea);
}

set_new_base(hwnd, fp, base_line, pbasepos)
 HWND hwnd;
 int fp, base_line;
 long *pbasepos;
```

```
{
 int i, last_base_line, last_line;
 long lastbasepos, lastpos;
 char buffer[MAXLENGTH];

 last_base_line = GetWindowWord(hwnd,6);
 lastbasepos = GetWindowLong(hwnd,8);
 last_line = GetWindowWord(hwnd, 2);
 if (base_line > last_base_line)
  {
   if ((base_line - last_base_line) <= (last_line - base_line))
    {
     lseek(fp, lastbasepos, Ø);
     for (i = Ø; i < (base_line - last_base_line); i++)
      {
        getline(buffer, MAXLENGTH, fp);
      }
     *pbasepos = lseek(fp, Ø1,1);
    }
   else
    {
     lastpos = lseek(fp, Ø1, 2);
     seekback(fp, last_line, lastpos, base_line, pbasepos);
    }
  }
 else
  {
   if (base_line <= (last_base_line - base_line))
    {
     lseek(fp, Ø1, Ø);
     for (i = Ø; i < base_line; i++)
      {
        getline(buffer, MAXLENGTH, fp);
      }
     *pbasepos = lseek(fp, Ø1,1);
    }
   else
    {
     if (base_line == last_base_line)
      {
       lseek(fp, lastbasepos, Ø);
       *pbasepos = lastbasepos;
      }
     else
      seekback(fp, last_base_line, lastbasepos, base_line, pbasepos);
    }
  }
}

seekback(fp, current_base_line, current_base_pos,new_base_line, pbasepos)
 int fp, current_base_line, new_base_line;
 long current_base_pos, *pbasepos;

 {
  int i;
  long new_pos;
  static long pos[1ØØØ];
  char buffer[MAXLENGTH];

  while (new_base_line < current_base_line)
   {i = Ø;
    if (current_base_pos <= 1ØØØl)
     {
      lseek(fp, Øl, Ø);
      for (i = Ø; i < new_base_line; i++)
       {
         getline(buffer, MAXLENGTH, fp);
       }
      *pbasepos = lseek(fp, Øl, 1);
      return;
     }
    lseek(fp, current_base_pos - 1ØØØl, Ø);
    getline(buffer,MAXLENGTH, fp);
    pos[Ø] = lseek(fp, Øl, 1);
    while(pos[i] < current_base_pos)
     {
      getline(buffer, MAXLENGTH, fp);
      i++;
      pos[i] = lseek(fp, Øl, 1);
     }
    current_base_line -= i;
    current_base_pos = pos[Ø];
   }
  i = new_base_line - current_base_line;
  *pbasepos = pos[i];
  lseek(fp, *pbasepos, Ø);
 }

vert_scroll (hwnd, jump_type, new_pos)

  HWND hwnd;
  WORD jump_type;
  int  new_pos;


  int  old_pos, page_length, min_pos, max_pos;
  RECT window_rect;

  GetScrollRange(hwnd, SB_VERT, (LPINT)&min_pos, (LPINT)&max_pos);
  GetClientRect(hwnd, (LPRECT)&window_rect);
  page_length = window_rect.bottom/font_sizing.tmHeight -
                window_rect.top/font_sizing.tmHeight;
  old_pos = GetScrollPos(hwnd, SB_VERT);
  switch (jump_type)
   {
     case SB_LINEUP:
          if (old_pos > min_pos)
             {
               SetScrollPos(hwnd, SB_VERT, old_pos-1, TRUE);
               ScrollWindow(hwnd, Ø, font_sizing.tmHeight, (LPRECT)NULL,
                            (LPRECT)NULL);
```

```
              }
            break;
        case SB_LINEDOWN:
            if (old_pos < max_pos)
              {
                SetScrollPos(hwnd, SB_VERT, old_pos+1, TRUE);
                ScrollWindow(hwnd, Ø, -font_sizing.tmHeight, (LPRECT)NULL,
                             (LPRECT)NULL);
              }
            break;
        case SB_PAGEUP:
            if (old_pos > min_pos)
              {
                if ((old_pos -= page_length) < min_pos)
                  {
                    old_pos = min_pos;
                  }
                SetScrollPos(hwnd, SB_VERT, old_pos, TRUE);
                InvalidateRect(hwnd, (LPRECT)NULL, FALSE);
              }
            break;
        case SB_PAGEDOWN:
            if (old_pos < max_pos)
              {
                if ((old_pos += page_length) > max_pos)
                  {
                    old_pos = max_pos;
                  }
                SetScrollPos(hwnd, SB_VERT, old_pos, TRUE);
                InvalidateRect(hwnd, (LPRECT)NULL, FALSE);
              }
            break;
        case SB_THUMBPOSITION:
            SetScrollPos(hwnd, SB_VERT, new_pos, TRUE);
            InvalidateRect(hwnd, (LPRECT)NULL, FALSE);
            break;
        case SB_THUMBTRACK:
            break;
        case SB_TOP:
            SetScrollPos(hwnd, SB_VERT, min_pos, TRUE);
            InvalidateRect(hwnd, (LPRECT)NULL, FALSE);
            break;
        case SB_BOTTOM:
            SetScrollPos(hwnd, SB_VERT, max_pos, TRUE);
            InvalidateRect(hwnd, (LPRECT)NULL, FALSE);
            break;
        case SB_ENDSCROLL:
            break;
      }
}

change_file(window_handle, new_file)
char *new_file;

{
  int num_cols, num_lines;
```

```
 close(GetWindowWord(window_handle, Ø));
 new_fp = open(new_file, O_RDONLY | O_BINARY);
 SetWindowWord(window_handle, Ø, new_fp);
 scanfile(new_fp, &num_lines, &num_cols);
 SetWindowWord(window_handle, 2, num_lines);
 SetWindowWord(window_handle, 4, num_cols);
 SetWindowWord(window_handle, 6, Ø);
 SetWindowLong(window_handle, 8, Ø1);
}

change_current_coa()

{
 int resp, i;
 char str[3Ø];
 RECT window_rect;
 char st1[8Ø], st2[8Ø],st3[8Ø];

 resp = MessageBox(current_figure, (LPSTR)"New COA indicated.  Review new COA?",
             (LPSTR)"CHANGE COA", MB_YESNO);
 if (resp == IDNO)
  return;
 current_f_coa = get_current_coa(fcoa, f_curvec);
 current_e_coa = get_current_coa(ecoa, e_curvec);
 for (i = Ø; i < no_of_open_files - 1; i++)
  {
   GetWindowRect(file_handle[i], (LPRECT)&window_rect);
   switch (i == Ø ? figflag : -figflag)
    {
      case 146:
      case 148:  case -138:
      case 137:
           strcpy(st1,FRIENDLY_FILE_NAME);
           if (no_of_open_files == 3)
            sprintf(st2,"%s%c%d",st1,'s',current_f_coa);
           else
            sprintf(st2,"%s%d",st1,current_f_coa);
           sprintf(st3, "BLUE COA #%d",current_f_coa);
           SetWindowText(file_handle[i], (LPSTR)st3);
           break;
      case 136:
      case 138:  case -148:
      case 147:
           strcpy(st1,ENEMY_FILE_NAME);
           if (no_of_open_files == 3)
            sprintf(st2,"%s%c%d",st1,'s',current_e_coa);
           else
            sprintf(st2,"%s%d",st1,current_e_coa);
           sprintf(st3, "RED COA #%d",current_e_coa);
           SetWindowText(file_handle[i], (LPSTR)st3);
           break;
    }
   change_file(file_handle[i], st2);
   InvalidateRect(file_handle[i], (LPRECT)NULL, TRUE);
   SendMessage(file_handle[i], WM_SIZE, SIZENORMAL,
                ((LONG)(window_rect.bottom - window_rect.top) << 16) |
                window_rect.right - window_rect.left);
  }
 switch (no_of_open_files)
  {
    case 1:
         SendMessage(current_figure,WM_COMMAND,DLG_BUTTON1,(LONG)(LPSTR)NULL);
    case 2:
         PostMessage(current_figure,WM_COMMAND,DLG_BUTTON,(LONG)(LPSTR)NULL);
         break;
  }
 redo_screen();
}

set_box(hwnd, lparam, no_of_boxes,coors,curvec,num_sets)

 HWND hwnd;
 LONG lparam;
 int no_of_boxes[];
 RECT coors[][6];
 unsigned char curvec[];

 {
  POINT point;
  POINT screen;
  RECT rect;
  int i, j, k, boxflag, base_line, base_col;

  boxflag = FALSE;
  point = MAKEPOINT(lparam);
  base_line = GetScrollPos(hwnd, SB_VERT);
  base_col = GetScrollPos(hwnd, SB_HORZ);
  screen.x =
     point.x / font_sizing.tmMaxCharWidth + base_col + 1;
  screen.y =
     point.y / font_sizing.tmHeight + base_line + 1;
  i = Ø;
  while (i < num_sets)
   {
    for(j = Ø; j < no_of_boxes[i]; j++)
     {
       if (((screen.x >= coors[i][j].left) &&
          (screen.x <= coors[i][j].right))  &&
          ((screen.y >= coors[i][j].top)  &&
          (screen.y <= coors[i][j].bottom)))
         {
          boxflag = TRUE;
          break;
         }
     }
     if (boxflag == TRUE)
      break;
     else i++;
   }
  if (boxflag == TRUE)
```

```
   {
    if ((curvec[i] & Øx8Ø) == Ø)
     {
      if (curvec[i])
         {
           for (k = Ø; !(curvec[i] & (1 << k)); k++)
            {
            }
           rect.left = (coors[i][k].left-base_col)*font_sizing.tmMaxCharWidth-1;
           rect.top = (coors[i][k].top-base_line)*font_sizing.tmHeight-4;
           rect.right = (coors[i][k].right-base_col)*font_sizing.tmMaxCharWidth-
6;
           rect.bottom = (coors[i][k].bottom-base_line)*font_sizing.tmHeight-6;
           InvalidateRect(hwnd,(LPRECT)&rect,FALSE);
         }
      curvec[i] = (Øx Ø1 << j);
     }
    else
     {
      curvec[i] ^= (ØxØ1 << j);
     }
    rect.left = (coors[i][j].left-base_col)*font_sizing.tmMaxCharWidth-1;
    rect.top = (coors[i][j].top-base_line)*font_sizing.tmHeight-4;
    rect.right = (coors[i][j].right-base_col)*font_sizing.tmMaxCharWidth-6;
    rect.bottom = (coors[i][j].bottom-base_line)*font_sizing.tmHeight-6;
    InvalidateRect(hwnd,(LPRECT)&rect,FALSE);
   }
 }

count_bits(n)
unsigned char n;
{
 int i, tot;

 tot = Ø;
 for (i = Ø; i < 6; i++)
  {
   tot += (n >> i) & ØxØ1;
  }
 return tot;
}

find_max(j)
int j[];
{
 int i;

 i = Ø;
 if (j[1] > j[Ø]) i = 1;
 if (j[2] > j[i]) i = 2;
 return (i+1);
}
```

```
#include "intacval.h"

unsigned char ecoa[][3] =
        {
         0,         0,     0,                      /*1.1*/
         0x01,  0x01,  0x01,                       /*1.2*/
         0x02,  0x02,  0x02,                       /*2.1*/
         0x15,  0x15,  0x15,                       /*3.1*/
         0x07,  0x08,  0x10,                       /*3.2*/
         0x01,  0x01,  0x01,                       /*3.3*/
         0x03,  0x04,  0x04,                       /*3.4*/
         0x04,  0x01,  0x02,                       /*3.5*/
         0x19,  0x12,  0x14,                       /*4.1*/
         0x07,  0x00,  0x07,                       /*4.2*/
         0x01,  0x01,  0x01,                       /*4.3*/
         0x02,  0x04,  0x01,                       /*4.4*/
         0x02,  0x01,  0x02,                       /*4.5*/
         0x01,  0x01,  0x01,                       /*4.6*/
         0x04,  0x04,  0x01,                       /*5.1*/
         0x04,  0x04,  0x01,                       /*5.2*/
         0x02,  0x04,  0x01,                       /*5.3*/
         0x02,  0x04,  0x01,                       /*5.4*/
         0x01,  0x04,  0x01,                       /*5.5*/
         0x02,  0x04,  0x02,                       /*5.6*/
         0x02,  0x02,  0x02,                       /*5.7*/
         0x02,  0x02,  0x02,                       /*5.8*/
         0,0,0,                                    /*5.9*/
         0,0,0                                     /*5.10*/
        };

unsigned char fcoa[][3] =
        {
         0x01,  0x01,  0x01,                       /*1.1*/
         0x00,  0x00,  0x00,                       /*1.2*/
         0x02,  0x01,  0x02,                       /*2.1*/
         0x15,  0x1d,  0x15,                       /*3.1*/
         0x03,  0x30,  0x0c,                       /*3.2*/
         0x01,  0x01,  0x01,                       /*3.3*/
         0x01,  0x02,  0x02,                       /*3.4*/
         0x02,  0x01,  0x01,                       /*3.5*/
         0x1f,  0x15,  0x13,                       /*4.1*/
         0x07,  0x00,  0x07,                       /*4.2*/
         0x01,  0x01,  0x01,                       /*4.3*/
         0x04,  0x02,  0x01,                       /*4.4*/
         0x01,  0x01,  0x02,                       /*4.5*/
         0x01,  0x01,  0x01,                       /*4.6*/
         0x01,  0x01,  0x04,                       /*5.1*/
         0x01,  0x04,  0x01,                       /*5.2*/
         0x02,  0x02,  0x04,                       /*5.3*/
         0x02,  0x04,  0x01,                       /*5.4*/
         0x01,  0x02,  0x04,                       /*5.5*/
         0x01,  0x02,  0x01,                       /*5.6*/
         0x02,  0x02,  0x02,                       /*5.7*/
         0x01,  0x01,  0x01,                       /*5.8*/
         0,0,0,                                    /*5.9*/
         0,0,0                                     /*5.10*/
```

```
     };

unsigned char e_curvec[] = {
    Ø, Øx01, ØxØ2, Øx95,ØxØ4, ØxØ1,
    ØxØ2, ØxØ4, Øx99, ØxØ4, ØxØ1, ØxØ2,
    ØxØ2, Ø, ØxØ4, ØxØ4, ØxØ2, ØxØ2,
    ØxØ1, Ø, Ø, Ø, Ø, Ø
                  };

unsigned char f_curvec[] = {
    ØxØ1, ØxØØ, ØxØ2, Øx95,ØxØ1, ØxØ1,
    ØxØ1, ØxØ2, Øx86, ØxØ1, ØxØ1, ØxØ1,
    ØxØ1, Ø, ØxØ4, ØxØ4, ØxØ2, ØxØ2,
    ØxØ1, Ø, Ø, Ø, Ø, Ø
                  };

unsigned char e_defvec[] = {
    Ø, ØxØ1, ØxØ2, Øx95,ØxØ4, ØxØ1,
    ØxØ2, ØxØ4, Øx99, ØxØ4, ØxØ1, ØxØ2,
    ØxØ2, Ø, ØxØ4, ØxØ4, ØxØ2, ØxØ2,
    ØxØ1, Ø, Ø, Ø, Ø, Ø
                  };

unsigned char f_defvec[] = {
    ØxØ1, ØxØØ, ØxØ2, Øx95,ØxØ1, ØxØ1,
    ØxØ1, ØxØ2, Øx86, ØxØ1, ØxØ1, ØxØ1,
    ØxØ1, Ø, ØxØ4, ØxØ4, ØxØ2, ØxØ2,
    ØxØ1, Ø, Ø, Ø, Ø, Ø
                  };

int e_no_of_boxes[] =
      {
       Ø,                              /*1.1*/
       1,                              /*1.2*/
       2,                              /*2.1*/
       5,                              /*3.1*/
       6,                              /*3.2*/
       1,                              /*3.3*/
       4,                              /*3.4*/
       3,                              /*3.5*/
       5,                              /*4.1*/
       6,                              /*4.2*/
       1,                             -/*4.3*/
       4,                              /*4.4*/
       3,                              /*4.5*/
       3,                              /*4.6*/
       3,                              /*5.1*/
       3,                              /*5.2*/
       3,                              /*5.3*/
       3,                              /*5.4*/
       3,                              /*5.5*/
       3,                              /*5.6*/
       3,                              /*5.7*/
       3,                              /*5.8*/
       3,                              /*5.9*/
       3,                              /*5.1Ø*/
    };

int f_no_of_boxes[] =
    {
     1,     /*1.1*/
     Ø,     /*1.2*/
     2,     /*2.1*/
     5,     /*3.1*/
     6,     /*3.2*/
     1,     /*3.3*/
     4,     /*3.4*/
     3,     /*3.5*/
     5,     /*4.1*/
     6,     /*4.2*/
     1,     /*4.3*/
     4,     /*4.4*/
     3,     /*4.5*/
     3,     /*4.6*/
     3,     /*5.1*/
     3,     /*5.2*/
     3,     /*5.3*/
     3,     /*5.4*/
     3,     /*5.5*/
     3,     /*5.6*/
     3,     /*5.7*/
     3,     /*5.8*/
     3,     /*5.9*/
     3      /*5.1Ø*/
    };

RECT e_coors[][6] =
    {
/*   Ø,        1.1*/
        {
          Ø,  Ø,  Ø,  Ø
        },
/*   1,        1.2*/
        {
          3, 16, 14, 19
        },
/*   2,        2.1*/
        {
         31, 11, 45, 16,
         47, 11, 56, 13
        },
/*   5,        3.1*/
        {
         58, 12, 68, 15,
         7Ø, 12, 84, 15,
         54, 17, 68, 2Ø,
         7Ø, 17, 8Ø, 21,
         63, 22, 75, 25
        },
/*   6,        3.2*/
        {
         87, 12, 93, 14,
                 95, 12,102, 14,
                 86, 16, 93, 18,
                 95, 16,103, 18,
                 85, 20, 93, 22,
                 95, 20,102, 22
               },
/*     1,        3.3*/
               {
                105,  7,121, 10
               },
/*     4,        3.4*/
               {
                119, 13,129, 15,
                131, 13,137, 15,
                121, 17,129, 19,
                131, 17,137, 19
               },
/*     3,        3.5*/
               {
                139, 12,145, 14,
                147, 12,155, 14,
                143, 15,149, 17
               },
/*     5,        4.1*/
               {
                157, 11,169, 14,
                171, 11,181, 14,
                157, 16,169, 20,
                171, 16,181, 19,
                165, 21 174, 25
               },
/*     6,        4.2*/
               {
                184, 11,190, 13,
                192, 11,199, 13,
                183, 15,190, 17,
                192, 15,199, 17,
                183, 19,190, 21,
                192, 19,198, 21
               },
/*     1,        4.3*/
               {
                201,  6,214, 9
               },
/*     4,        4.4*/
               {
                212, 12,222, 14,
                224, 12,230, 14,
                214, 16,222, 18,
                224, 16,230, 18
               },
/*     3,        4.5*/
               {
                232, 11,238, 13,
                240, 11,248, 13,
                250, 11,256, 13
```

```
        },
/*   3,        4.6*/
        {
         258, 11,267, 15,
         269, 11,281, 15,
         283, 11,287, 13
        },
/*   3,        5.1*/
        {
         289, 13,296, 16,
         298, 13,3Ø4, 15,
         3Ø6, 13,313, 16
        },
/*   3,        5.2*/
        {
         315, 13,322, 16,
         324, 13,33Ø, 15,
         332, 13,339, 16
        },
/*   3,        5.3*/
        {
         341, 12,348, 15,
         35Ø, 12,356, 14,
         358, 12,365, 15
        },
/*   3,        5.4*/
        {
         367, 12,374, 15,
         376, 12,382, 14,
         384, 12,391, 15
        },
/*   3,        5.5*/
        {
         393, 12,4ØØ, 15,
         4Ø2, 12,4Ø8, 14,
         41Ø, 12,417, 15
        },
/*   3,        5.6*/
        {
         289, 22,296, 25,
         298, 22,3Ø4, 25,
         3Ø6, 22,313, 25
        },
/*   3,        5.7*/
        {
         315, 22,322, 25,
         324, 22,33Ø, 24,
         332, 22,339, 25
        },
/*   3,        5.8*/
        {
         341, 23,348, 26,
         35Ø, 23,356, 25
         358, 23,365, 26
        },
/*   3,        5.9*/
```

```
        {
         367, 23,374, 26,
         376, 23,382, 25,
         384, 23,391, 26
        },
/*   3       5.1Ø*/
        {
         393, 23,4ØØ, 26,
         4Ø2, 23,4Ø9, 25,
         411, 23,418, 26
        }
    };

RECT f_coors[][6] =
    {
/*   1,      1.1*/
        {
           1,  7,  9,  9
        },
/*   5,      1.2*/
        {
           3, 11, 14, 14,
          16, 11, 29, 13,
           3, 16, 14, 19,
          16, 16, 24, 19,
          11, 2Ø, 19, 22
        },
/*   2,      2.1*/
        {
          31, 11, 45, 16,
          47, 11, 56, 13
        },
/*   5,      3.1*/
        {
          58, 12, 68, 15,
          7Ø, 12, 84, 15,
          54, 17, 68, 2Ø,
          7Ø, 17, 8Ø, 21,
          63, 22, 75, 25
        },
/*   6,      3.2*/
        {
          87, 12, 93, 14,
          95, 12,1Ø2, 14,
          86, 16, 93, 18,
          95, 16,1Ø3, 18,
          85, 2Ø, 93, 22,
          95, 2Ø,1Ø2, 22
        },
/*   1,      3.3*/
        {
         1Ø5,  7,121, 1Ø
        },
/*   4,      3.4*/
        {
         119, 13,129, 15,
          131,  13,137,  15,
          121,  17,129,  19,
          131,  17,137,  19
         },
/*   3,       3.5*/
         {
          139,  12,145,  14,
          147,  12,155,  14,
          143,  15,149,  17
         },
/*   5,       4.1*/
         {
          157,  11,169,  14,
          171,  11,181,  14,
          157,  16,169,  2Ø,
          171,  16,181,  19,
          165,  21,174,  25
         },
/*   6,       4.2*/
         {
          184,  11,19Ø,  13,
          192,  11,199,  13,
          183,  15,19Ø,  17,
          192,  15,199,  17,
          183,  19,19Ø,  21,
          192,  19,198,  21
         },
/*   1,       4.3*/
         {
          2Ø1,   6,214,   9
         },
/*   4,       4.4*/
         {
          212,  12,222,  14,
          224,  12,23Ø,  14,
          214,  16,222,  18,
          224,  16,23Ø,  18
         },
/*   3,       4.5*/
         {
          232,  11,238,  13,
          24Ø,  11,248,  13,
          25Ø,  11,256,  13
         },
/*   3,       4.6*/
         {
          258,  11,267,  15,
          269,  11,281,  15,
          283,  11,287,  13
         },
/*   3,       5.1*/
         {
          289,  13,296,  16,
          298,  13,3Ø4,  15,
          3Ø6,  13,313,  16
         },
/*    3,       5.2*/
          {
           315, 13,322, 16,
           324, 13,33Ø, 15,
           332, 13,339, 16
          },
/*    3,       5.3*/
          {
           341, 12,348, 15,
           35Ø, 12,356, 14,
           358, 12,365, 15
          },
/*    3,       5.4*/
          {
           367, 12,374, 15,
           376, 12,382, 14,
           384, 12,391, 15
          },
/*    3,       5.5*/
          {
           393, 12,4ØØ, 15,
           4Ø2, 12,4Ø8, 14,
           41Ø, 12,417, 15
          },
/*    3,       5.6*/
          {
           289, 22,296, 25,
           298, 22,3Ø4, 25,
           3Ø6, 22,313, 25
          },
/*    3,       5.7*/
          {
           315, 22,322, 25,
           324, 22,33Ø, 24,
           332, 22,339, 25
          },
/*    3,       5.8*/
          {
           341, 23,348, 26,
           35Ø, 23,356, 25,
           358, 23,365, 26
          },
/*    3,       5.9*/
          {
           367, 23,374, 26,
           376, 23,382, 25,
           384, 23,391, 26
          },
/*    3        5.1Ø*/
          {
           393, 23,4ØØ, 26,
           4Ø2, 23,4Ø9, 25,
           411, 23,418, 26
          }
      };
```

```
RECT e_grays[] =
 {
                                         /*1.1*/
             1,  7,  9,  9,
                                         /*1.2*/
             3, 11, 14, 14,
            16, 11, 29, 13,
            16, 16, 24, 19,
            11, 2Ø, 19, 22
 };

RECT f_grays[] =
 {
                                         /*1.2*/
             3, 11, 14, 14,
            16, 11, 29, 13,
             3, 16, 14, 19,
            16, 16, 24, 19,
            11, 2Ø, 19, 22
 };

#define NUM_GRAYS (sizeof(e_grays)/sizeof(e_grays[Ø]))
#define NUM_SETS (sizeof(e_no_of_boxes)/sizeof(e_no_of_boxes[Ø]))

long pascal HandleBoxesWindow (hWnd, wMsg, wParam, lParam)

  HWND hWnd;
  WORD wMsg, wParam;
  LONG lParam;

{
  LONG retval;
  int  new_horz_max, new_vert_max;
  int fp, num_lines, num_cols;
  int new_i;

  switch (wMsg)
   {
     case WM_SYSCOMMAND:
          switch (wParam & Øxfff0)
           {
        case SC_KEYMENU:
         if (lParam == 9)
          {
           return(DefWindowProc (hWnd, wMsg, SC_NEXTWINDOW, lParam));
          }
         else
          {
           return(DefWindowProc (hWnd, wMsg, wParam, lParam));
          }
         break;
        case SC_CLOSE:
         delete_one_child(hWnd,&new_i);
         destroyed_flag = TRUE;
         if (figflag >= 13Ø)
          change_window_pos(hWnd,new_i);
          return(DefWindowProc (hWnd, wMsg, wParam, lParam));
         default:
          return(DefWindowProc (hWnd, wMsg, wParam, lParam));
          break;
       }
      break;
     case WM_LBUTTONDOWN:
          if (figflag >= 140)
          {
           set_box(hWnd, lParam,f_no_of_boxes,f_coors,f_curvec.NUM_SETS);
          }
          else
          {
           set_box(hWnd, lParam,e_no_of_boxes,e_coors,e_curvec,NUM_SETS);
          }
          if ((current_f_coa != get_current_coa(fcoa, f_curvec)) ||
              (current_e_coa != get_current_coa(ecoa, e_curvec)))
            {
              change_current_coa();
            }
          return(1L);
          break;
     case WM_PAINT:
         if (figflag >= 140)
           paintboxes (hWnd, (LPPAINTSTRUCT)lParam,
            f_curvec,f_no_of_boxes,f_coors,f_grays);
         else
           paintboxes (hWnd, (LPPAINTSTRUCT)lParam,
            e_curvec,e_no_of_boxes,e_coors,e_grays);
         return(1L);
          break;
     case WM_HSCROLL:
          side_scroll(hWnd, wParam, LOWORD(lParam));
          return(1L);
          break;
     case WM_VSCROLL:
          vert_scroll(hWnd, wParam, LOWORD(lParam));
          return(1L);
          break;
     case WM_SIZE:
          retval = DefWindowProc(hWnd, wMsg, wParam, lParam);
          switch (wParam)
           {
             case SIZEFULLSCREEN:
             case SIZENORMAL:
                  new_horz_max = GetWindowWord(hWnd, 4);
                  new_vert_max = GetWindowWord(hWnd, 2);
                  new_horz_max -= LOWORD(lParam)/font_sizing.tmMaxCharWidth;
                  new_vert_max -= HIWORD(lParam)/font_sizing.tmHeight;
                  if (new_horz_max < 0) new_horz_max = 0;
                  if (new_vert_max < 0) new_vert_max = 0;
                  if ((GetScrollPos(hWnd, SB_HORZ) > new_horz_max)
                    ||(GetScrollPos(hWnd, SB_VERT) > new_vert_max))
                     {
                       SetScrollPos(hWnd, SB_HORZ, new_horz_max, FALSE);
                       SetScrollPos(hWnd, SB_VERT, new_vert_max, FALSE);
```

```
                        InvalidateRect(hWnd, (LPRECT)NULL, FALSE);
                      }
                   SetScrollRange(hWnd, SB_HORZ, Ø, new_horz_max, TRUE);
                   SetScrollRange(hWnd, SB_VERT, Ø, new_vert_max, TRUE);
                   break;
              default:
                   break;
            }
          return(retval);
          break;
     case WM_DESTROY:
          if (figflag <= 14Ø)
           update_opposing_forces(e_curvec,f_curvec);
          else
           update_opposing_forces(f_curvec, e_curvec);
          fp = GetWindowWord(hWnd, Ø);
          close(fp);
          if (destroyed_flag == FALSE)
           delete_one_child(hWnd,&new_i);

          destroyed_flag = FALSE;
          return(DefWindowProc (hWnd, wMsg, wParam, lParam));
     case WM_CREATE:
          SetWindowWord(hWnd, Ø, new_fp);
          scanfile(new_fp, &num_lines, &num_cols);
          SetWindowWord(hWnd, 2, num_lines);
          SetWindowWord(hWnd, 4, num_cols);
          SetWindowWord(hWnd, 6, Ø);
          SetWindowLong(hWnd, 8, Ø1);
          BringWindowToTop(hWnd);
          return(DefWindowProc (hWnd, wMsg, wParam, lParam));
     default:
          return(DefWindowProc (hWnd, wMsg, wParam, lParam));
          break;
   }
}


paintboxes (hwnd, lpArea,curvec,no_of_boxes,coors,grays)

  HWND hwnd;
  LPPAINTSTRUCT lpArea;
  unsigned char curvec[];
  int no_of_boxes[];
  RECT coors[][6], grays[];

{
  char buffer[MAXLENGTH + 5];
  char longbuffer[MAXLENGTH + 5];
  int  i, j, start_col, num_cols, base_col, top_edge;
  int  start_line, num_lines, base_line, last_base_line;
  int fp, n;
  long basepos, lastbasepos;
  char s[8Ø];
  RECT rect;
  HBRUSH gray_brush, color_brush;
```

```
base_col = GetScrollPos(hwnd, SB_HORZ);
base_line = GetScrollPos(hwnd, SB_VERT);
BeginPaint(hwnd, lpArea);
GetClientRect(hwnd, (LPRECT)&rect);
FillRect(lpArea->hdc, (LPRECT)&rect, GetStockObject(WHITE_BRUSH));
gray_brush = CreateSolidBrush(ØxcØcØcØ);
for (i = Ø; i < NUM_GRAYS; i++)
 {
   rect.left = (grays[i].left-base_col)*font_sizing.tmMaxCharWidth-1;
   rect.top = (grays[i].top-base_line)*font_sizing.tmHeight-4;
   rect.right = (grays[i].right-base_col)*font_sizing.tmMaxCharWidth-6;
   rect.bottom = (grays[i].bottom-base_line)*font_sizing.tmHeight-6;
   FillRect(lpArea->hdc, (LPRECT)&rect, gray_brush);
 }
DeleteObject(gray_brush);
color_brush = CreateSolidBrush(figflag < 14Ø ? ØxØØØØffL : ØxffffØØL);
for (i = Ø; i< NUM_SETS; i++)
 {
  for(j = Ø; j < no_of_boxes[i]; j++)
    {
     if (curvec[i] & (ØxØ1 << j))
      {
       invert_box(i,j,lpArea,base_col, base_line,coors,color_brush);
      }
    }
 }
DeleteObject(color_brush);
SelectObject(lpArea->hdc, GetStockObject(OEM_FIXED_FONT));
start_col = lpArea->rcPaint.left/font_sizing.tmMaxCharWidth;
num_cols = lpArea->rcPaint.right/font_sizing.tmMaxCharWidth-start_col+1;
top_edge = (lpArea->rcPaint.top/font_sizing.tmHeight)*font_sizing.tmHeight;
start_line = lpArea->rcPaint.top/font_sizing.tmHeight;
num_lines = lpArea->rcPaint.bottom/font_sizing.tmHeight-start_line+1;
fp = GetWindowWord(hwnd, Ø);
set_new_base(hwnd, fp, base_line, &basepos);
basepos = lseek(fp, Øl, 1);
SetWindowLong(hwnd, 8, basepos);
SetWindowWord(hwnd, 6, base_line);
for (i = Ø; i < start_line; i++)
 {
   getline(buffer, MAXLENGTH, fp);
 }
SetBkMode(lpArea->hdc, TRANSPARENT);
for (i = Ø; i < num_lines; i++)
 {
   getline(buffer, MAXLENGTH, fp);
   initvec(longbuffer, MAXLENGTH);
   transbuffer(buffer, longbuffer);
   TextOut(lpArea->hdc, start_col*font_sizing.tmMaxCharWidth,
           (i  + start_line)* font_sizing.tmHeight,
           (LPSTR)(longbuffer +
           base_col+start_col), num_cols);
 }
SetBkMode(lpArea->hdc, OPAQUE);
EndPaint(hwnd, lpArea);
```

```
display_oav(n)
 int n;
 {
  no_of_open_files++;
  new_fp = open("oav", O_BINARY);
  file_handle[n] = CreateWindow((LPSTR)"BOXES",
       (figflag < 14Ø ? (LPSTR)"RED KNOWLEDGE BASE" : (LPSTR)"BLUE KNOWLEDGE BAS
E"),
       WS_CHILD|WS_SYSMENU|WS_CLIPSIBLINGS|WS_CAPTION|WS_VISIBLE,
       1ØØ,13Ø, 1, 1, main_window, NULL,
       intacval_instance,(LPSTR)NULL);
 }

invert_box(i,j,lpArea,base_col,base_line, coors, color_brush)
 int i,j;
 LPPAINTSTRUCT lpArea;
 int base_col,base_line;
 RECT coors[][6];
 HBRUSH color_brush;

 {
  RECT screen;
  static int fp;
  screen.left =  ((coors[i][j]).left - base_col)
                                   * font_sizing.tmMaxCharWidth-1;
  screen.top =   ((coors[i][j]).top - base_line)
                                   * font_sizing.tmHeight-4;
  screen.right = ((coors[i][j]).right - base_col)
                                   * font_sizing.tmMaxCharWidth-6;
  screen.bottom = ((coors[i][j]).bottom - base_line)
                                   * font_sizing.tmHeight-6;
  FillRect(lpArea->hdc, (LPRECT)&screen, color_brush);
 }


 ide_scroll (hwnd, jump_type, new_pos)

  HWND hwnd;
  WORD jump_type;
  int  new_pos;

 {
  int  old_pos, page_width, min_pos, max_pos;
  RECT window_rect;

  GetScrollRange(hwnd, SB_HORZ, (LPINT)&min_pos, (LPINT)&max_pos);
  GetClientRect(hwnd, (LPRECT)&window_rect);
  page_width = window_rect.right/font_sizing.tmMaxCharWidth-
               window_rect.left/font_sizing.tmMaxCharWidth;
  old_pos = GetScrollPos(hwnd, SB_HORZ);
  switch (jump_type)
   {
        case SB_LINEUP:
             if (old_pos > min_pos)
                {
                  SetScrollPos(hwnd, SB_HORZ, old_pos-1, TRUE);
                  ScrollWindow(hwnd, font_sizing.tmMaxCharWidth, Ø, (LPRECT)NULL,
                                  (LPRECT)NULL);
                }
             break;
        case SB_LINEDOWN:
             if (old_pos < max_pos)
                {
                  SetScrollPos(hwnd, SB_HORZ, old_pos+1, TRUE);
                  ScrollWindow(hwnd, -font_sizing.tmMaxCharWidth, Ø, (LPRECT)NULL,
                                  (LPRECT)NULL);
                }
             break;
        case SB_PAGEUP:
             if (old_pos > min_pos)
                {
                  if ((old_pos -= page_width) < min_pos)
                     {
                        old_pos = min_pos;
                     }
                  SetScrollPos(hwnd, SB_HORZ, old_pos, TRUE);
                  InvalidateRect(hwnd, (LPRECT)NULL, FALSE);
                }
             break;
        case SB_PAGEDOWN:
             if (old_pos < max_pos)
                {
                  if ((old_pos += page_width) > max_pos)
                     {
                        old_pos = max_pos;
                     }
                  SetScrollPos(hwnd, SB_HORZ, old_pos, TRUE);
                  InvalidateRect(hwnd, (LPRECT)NULL, FALSE);
                }
             break;
        case SB_THUMBPOSITION:
             SetScrollPos(hwnd, SB_HORZ, new_pos, TRUE);
             InvalidateRect(hwnd, (LPRECT)NULL, FALSE);
             break;
        case SB_THUMBTRACK:
             break;
        case SB_TOP:
             SetScrollPos(hwnd, SB_HORZ, min_pos, TRUE);
             InvalidateRect(hwnd, (LPRECT)NULL, FALSE);
             break;
        case SB_BOTTOM:
             SetScrollPos(hwnd, SB_HORZ, max_pos, TRUE);
             InvalidateRect(hwnd, (LPRECT)NULL, FALSE);
             break;
        case SB_ENDSCROLL:
             break;
     }
}
```

```
get_current_coa(coa, curvec)
unsigned char curvec[], coa[][3];

{
 int i, j[3], k;

 j[Ø] = Ø; j[1] = Ø; j[2] = Ø;
 for (i = Ø; i < NUM_SETS; i++)
  {
   for(k= Ø;k < 3; k++)
    {
     j[k] += count_bits((unsigned char)(coa[i][k] & curvec[i]));
    }
  }
 return find_max(j);
}

open_file (vec)
 char *vec;

 {
  FILE *fp;
  int i,n;
  char filename[15];

  for (i = Ø; i < 15; i++)
   {
    if (vec[i] == ' ') break;
    filename[i] = vec[i];
   }
  filename[i] = NULL;
  fp = fopen(filename, "r");
  for (i = Ø; i < NUM_SETS; i++)
   {
    fscanf(fp, "%x",&n);
    e_curvec[i] = n;
   }
  for (i = Ø; i < NUM_SETS; i++)
   {
    fscanf(fp, "%x",&n);
    f_curvec[i] = n;
   }
  fclose(fp);
  strcpy(current_session, filename);
  strcpy(s, "Session has been opened as requested.");
  SetDlgItemText(current_figure, DLG_CLICK, (LPSTR)s);
  current_f_coa = get_current_coa(fcoa, f_curvec);
  current_e_coa = get_current_coa(ecoa, e_curvec);
 }

init_session()
 {
  int i;

  get_dir(filenames,&i);
  if (i >= 1Ø)
   {
  SendMessage(main_window,WM_COMMAND,MC_PREVIOUS_SESSION,(LONG)(LPSTR)NULL);
  strcpy(s, "A new session can be opened only after an old one is deleted.");
  SetDlgItemText(current_figure, DLG_CLICK, (LPSTR)s);
  return;
   }
  for(i = Ø; i < NUM_SETS; i++)
   {
    e_curvec[i] = e_defvec[i];
    f_curvec[i] = f_defvec[i];
   }
  strcpy(current_session, next_session);
  strcpy(s, "A new session has been opened as requested.");
  SetDlgItemText(current_figure, DLG_CLICK, (LPSTR)s);
 }

save_cur_session()
{
 int i;
 FILE *fp;

 fp = fopen(current_session, "w");
 for (i = Ø; i < NUM_SETS; i++)
  {
   fprintf(fp, "\n%x ",e_curvec[i]);
  }
 for (i = Ø; i < NUM_SETS; i++)
  {
   fprintf(fp, "\n%x ",f_curvec[i]);
  }
 fclose(fp);
}
```

```
#include        <stdio.h>

#define FALSE   Ø
#define TRUE   -1
#define OKAY    Ø
#define ERROR  -1

#define VDINIT  Ø
#define VDPLAY  1
#define VDSTOP  2
#define VDSTAT  3
#define VDDISP  4
#define VDFRAME 5

#define FWD     1
#define BWD     (-1)
#define TIMEOUT (-1)

/*EJECT*/

/***************************************************************/
/*                     PIONEER DISC COMMANDS                   */
/***************************************************************/

#define BIN_IN          Øx3D
#define BIN_OUT         ØxE7
#define DISPLAY         ØxF1
#define FAST_SPD        ØxEC
#define FRAME           ØxD3
#define MULTI_FWD       ØxF2
#define PLAY            ØxFD
#define SEARCH          ØxF7
#define SLOW_SPD        ØxED
#define STEP_FWD        ØxF6
#define STEP_BWD        ØxFE
#define STATUS          ØxD4
#define STOP            ØxFB
#define VID_ON          Øx1B
#define VID_OFF         Øx1C

#define STAT_DELAY      5       /* About half a second */

#define COM             3       /* 1 = COM1:, 2 = COM2:, etc */

#if (COM == 1)
#define PORT            Øx3F8   /* COM1: */
#endif
#if (COM == 2)
#define PORT            Øx2F8   /* COM2: */
#endif
#if (COM == 3)
#define PORT            Øx2E8   /* COM3: */
#endif
#define DATA_PORT       PORT
#define LOW_BAUD        PORT
#define HIGH_BAUD       (PORT+1)
```

```
#define INT_CNTRL       (PORT+1)
#define LINE_CNTRL      (PORT+3)
#define MODEM_CNTRL     (PORT+4)
#define LINE_STATUS     (PORT+5)
#define MODEM_STATUS    (PORT+6)

/*EJECT*/

        open_vdisc()
        /*=============================================*/
        /*   Initialize and start videodisc player      */
        /*=============================================*/
        {
            int         done, stat, tolerance;

            outp(INT_CNTRL,Øx00);
            outp(MODEM_CNTRL,Øx0f);
            outp(LINE_CNTRL,Øx80);
            outp(LOW_BAUD,Øx30);
            outp(HIGH_BAUD,Øx00);
            outp(LINE_CNTRL,Øx03);
            send(BIN_IN);
            send(BIN_OUT);
            done = FALSE;
            tolerance = 1;
            while (!done)
              {
                stat = getstat();
                switch (stat)
                  {
                    case Øx64:  send(STOP);      /* Playing */
                                break;
                    case TIMEOUT:
                                if (tolerance--)
                                  {
                                    send(BIN_IN);
                                    send(BIN_OUT);
                                    break;
                                  }
                    case Øx65:  done = TRUE;     /* Still frame */
                                break;
                    case Øx78:  send(PLAY);      /* Parked */
                                break;
                    default:    send(BIN_IN);
                                send(BIN_OUT);
                                break;
                  }
                wait(STAT_DELAY);
              }
            seek_vdisc(1L);
            send_num(ØL);
            send(DISPLAY);
            getstat();
            send(VID_ON);               /* Turn on video */
        }
```

```
/*EJECT*/

        seek_vdisc(fnum)
        /*==============================================*/
        /*    Searches for the specified frame number    */
        /*==============================================*/
        long fnum;
        {
            unsigned    getframe();
            static long old_fnum = -2;
            int  status;

            if (fnum-old_fnum == 1) {
              send(STEP_FWD);
            }
            else if (fnum-old_fnum == -1) {
              send(STEP_BWD);
            }
            else {
              send_num(fnum);
              send(SEARCH);
            }
            while (((status = getstat()) != Øx65) && (status != TIMEOUT))
                wait(STAT_DELAY);
            old_fnum = fnum;
        }
/*EJECT*/

        send_num(frame)
        /*==============================================*/
        /*      Send the frame number to the player.    */
        /*==============================================*/
        long frame;
        {
            static char uei[] = { Øx3F, ØxØF, Øx8F, Øx4F, Øx2F,
                                  ØxAF, Øx6F, Øx1F, Øx9F, Øx5F };
            char        out[5];
            int         d, i;

            i = -1;
            do
              {                             /* Translate frame number */
                d = frame % 1Ø;
                out[++i] = uei[d];
              }
            while ((frame /= 1Ø) > Ø);

            for (; i >= Ø; i--)             /* Reverse it */
                send(out[i]);
        }
/*EJECT*/

        getstat()
        /*==============================================*/
        /*      Get the status of the LD-V6ØØØ          */
        /*==============================================*/
        {
            int         stat;

            send(STATUS);
            stat = recv();
            return(stat);
        }

/*EJECT*/

        send(byte)
        /*==============================================*/
        /*    Send a byte to the videodisc player.      */
        /*==============================================*/
        unsigned char    byte;
        {
            while ((inp(LINE_STATUS) & Øx2Ø) == Ø)
             {
             }
            outp(DATA_PORT,byte);
        }

static  recv()
        /*==============================================*/
        /*  Receive one byte from the player. Return    */
        /*  TIMEOUT if a Rx timeout occurs.             */
        /*==============================================*/
        {
            int  temp;
            long timer;

            temp = TIMEOUT;
            timer = Øx18ØØØ;
            while (timer--)
             {
               if (inp(LINE_STATUS) & 1)
                 {
                   temp = inp(DATA_PORT);
                   break;
                 }
             }
            return(temp);
        }
```

```
#include "lvm.h"

lvm()
{
  int  x, y;
  static int firstpass = 1;

  if (inpw(Øx3eØ) == -1)
     {
       return;
     }
  hijack_mouse();
  if (firstpass)
     {
       firstpass = Ø;
       m_setcurs(screen_w/2, screen_h/2);
       setcurs(Ø, 1, screen_x+(screen_w >> 1), screen_y+(screen_h >> 1));
       curson();
       map_point(Ø, markx, marky, Ø);
       redo_screen();
     }
  gw_tracker((int(*)())Ø);
  while (m_posbut(&x, &y))
   {
   }
  return_mouse();
}
```

```
#include "parallax.h"
#include "lvm.h"

#define PI  3.141592653589793
#define RADIAN PI/180
#include "math.h"

#define sign( n ) ((n) < 0 ? -1 : 1)

clip_vectw(int, int, long, long, long, long);

draw_icon(icon_index)

long icon_index;     /* index of the icon within the data base */

{

/* The icon class is a bit mask where each bit corresponds to a particular
   class. Only one bit, of course, will be turned on for every icon.
   For example, an enemy radar station is 1, a SAM is 2, a command post is
   4, etc.   For each class there is a corresponding icon representation
   (box, triangle, etc.) which is loaded into a table (as it were) of
   images. The bit position turned on in the class indexes into this
   position. */

  register int class = icons[icon_index].class;
  register int classbit = 1 << class;

  long x, y;      /*  pixel location at which to display */
  unsigned char color;     /*  color  */

/* If this type of icon isn't being displayed, don't bother */

  if (!(classbit & show_mask)) return;

  if (class >= RED_UNIT)
    color = RED;    /* Enemy icons are red (icon templates are white) */
  else
    color = BLUE;  /* Friendly icons are blue */

/*
  Calculate the pixel position by converting the lat, long and centering
  the icon about the resultant x/y.
*/

  latlon_to_pix(icons[icon_index].lat, icons[icon_index].lon, &x, &y);
  x -= icontab[class & 7].width/2;
  y -= icontab[class & 7].height/2;

/*
  This routine draws the font corresponding to this icon;
*/
  gw_icon_image(&icontab[class & 7], color, (int)x, (int)y);
}

arrow(color, x1, y1, x2, y2, arrowhead)
```

```
unsigned char color;
long x1, y1, x2, y2;
int  arrowhead;   /* width of arrow head in pixels */

   int triangle[3][2];
   double angle = atan2((double)(y2-y1), (double)(x2-x1));

   mask(Øx3f);
   clip_vectw(3, color, x1, y1, x2, y2);
   if ((x2 > arrowhead) && (x2 < 2Ø48-arrowhead) && (y2 > arrowhead) &&
        (y2 < 2Ø48-arrowhead))
      {
        triangle[Ø][Ø] = x2;
        triangle[Ø][1] = y2;
        triangle[1][Ø] = x2-cos(angle-PI/4)*arrowhead;
        triangle[1][1] = y2-sin(angle-PI/4)*arrowhead;
        triangle[2][Ø] = x2-cos(angle+PI/4)*arrowhead;
        triangle[2][1] = y2-sin(angle+PI/4)*arrowhead;
        poly(color, 3, triangle);
      }
   mask(Øxff);
}

#define METER_NAUT 185ØØL   /* 18.5 km. per 1Ø nautical miles */

clip_aoi()
{
   long x1, y1, x2, y2;

   latlon_to_pix(aoi_lat1, aoi_lon1, &x1, &y1);
   latlon_to_pix(aoi_lat2, aoi_lon2, &x2, &y2);
   clip((int)x1, (int)y1, (int)x2, (int)y2);
}

unclip_aoi()

   clipd();


#include "mouse.h"

get_aoi(x, y)

int x, y;


   int b, x1, y1, oldx, oldy;

/*
   Wait till the user releases the button that invoked the function
*/

   info("Release the button, please");
```

```
  while (m_posbut(&oldx, &oldy));
  m_setcurs(x, y);

/*
  Set up a circular cursor and set the limits of the cursor to be
  the size of the available map frame.
*/
  cursoff();                          /* set up circular cursor */
  setcurs(CRCUR, 1, screen_x+x, screen_y+y);
  curson();

  m_hminmax(map->curx, map->curx+map->width-1);
  m_vminmax(map->cury, map->cury+map->height-1);

  info("Move to lower left corner of box and click mouse button.");

  while (!m_posbut(&x, &y)) {       /* move ordinary cursor until he presses */
    if (oldx != x !! oldy != y) {
      movcurs(screen_x+x, screen_y+y);
      oldx = x;
      oldy = y;
    }
  }
  while (m_posbut(&x, &y)) {      /* move it until he releases. */
    if (oldx != x !! oldy != y) {
      movcurs(screen_x+x, screen_y+y);
      oldx = x;
      oldy = y;
    }
  }

  info("Move to upper right hand corner and click mouse button.");

  cursoff();
  rubberband(BOXO, RED, x, y, &x1, &y1);
  setcurs(XHCUR, 1, screen_x+x1, screen_y+y1);      /* Reset the cursor */
  curson();
  info("New area of interest has been set.");
  aoi_x1 = min(x, x1);
  aoi_y1 = min(y, y1);
  aoi_x2 = max(x, x1);
  aoi_y2 = max(y, y1);
  pix_to_latlon((long)aoi_x1, (long)aoi_y1, &aoi_lat1, &aoi_lon1);
  pix_to_latlon((long)aoi_x2, (long)aoi_y2, &aoi_lat2, &aoi_lon2);
  m_hminmax(8, screen_w-8);
  m_vminmax(8, screen_h-8);


draw_aoi()
{
  mask(Øx3f);
  boxo(RED, aoi_x1, aoi_y1, aoi_x2, aoi_y2);
  mask(Øxff);
```

```
}

show_terrain ()

{
  static struct
   {
     char color;
     int  num_points;
     struct POINT
      {
        long lat, lon;
      } point[14];
   } terrain[] =
    {
      {
        0x9e, 10, 545020, 164555, 543345, 161027, 540517, 141310,
                  541703, 133458, 541843, 131025, 544343, 133428,
                  540336, 103740, 543025, 110213, 543705, 101023,
                  544702, 100456
      },
      {
        0x44, 14, 543345, 184029, 505304, 181556, 510920, 162122,
                  511443, 114552, 520104, 102645, 512154,  70210,
                  534318,  71305, 540014,  91833, 543525, 101550,
                  542765, 105929, 540155, 104307, 543205, 124836,
                  540517, 141026, 544702, 165122
      },
      {
        0xa2, 8, 505304, 181556, 510920, 162122, 511443, 114552,
                 520104, 102645, 522154,  70210, 475930,  71548,
                 491140, 155405, 475540, 185935
      },
      {
        0xd4, 6, 475540, 185935, 491140, 155405, 475930,  71548,
                 463348,  74305, 464340, 131025, 460004, 184557
      },
      {
        0x00, 4, 522541, 112646, 523425, 115646, 522026, 115930,
                 520953, 112646
      },
      {
        0x00, 6, 523241, 130214, 524821, 125931, 525517,-131309,
                 525149, 133458, 523755, 134026, 522211, 131553
      },
      {
        0x00, 4, 535510,  92117, 535832,  94550, 534318, 100456,
                 533123,  94306
      },
      {
        0x00, 7, 474953, 114024, 475930, 112718, 481256, 111835,
                 482423, 113213, 482229, 115119, 481101, 120214,
                 475735, 115119
      },
      {
        0x00, 4, 502546, 140459, 150916, 134553, 495430, 141026,
                  5Ø1256, 134Ø26
        },
        {
          Øx00, 4, 491717,  9Ø739, 48583Ø,  93212, 484517,  91Ø22,
                  49ØØ23,  84549
        }
      };
#define NUM_TERRAINS ((sizeof terrain)/(sizeof terrain[Ø]))
  struct POINT *point;
  struct
   {
     int  x, y;
   } coords[14];
  int  i, j;
  static char first_pass = 1;
  long x, y;
  char fits;

  if (first_pass)
    {
      first_pass = Ø;
      for (i = Ø; i < NUM_TERRAINS; i++)
       {
         for (j = Ø; j < terrain[i].num_points; j++)
          {
            terrain[i].point[j].lat = merge_deg(terrain[i].point[j].lat);
            terrain[i].point[j].lon = merge_deg(terrain[i].point[j].lon);
          }
       }
    }
  if (!(show_mask & (1 << TERRAIN)))
    {
      return;
    }
  for (i = Ø; i < NUM_TERRAINS; i++)
   {
     fits = 1;
     point = terrain[i].point;
     for (j = Ø; j < terrain[i].num_points; j++)
      {
        latlon_to_pix(point->lat, point->lon, &x, &y);
        if ((x < Ø) || (x >= screen_w) || (y < Ø) || (y >= screen_h))
          {
            fits = Ø;
            break;
          }
        coords[j].x = x;
        coords[j].y = y;
        point++;
      }
     if (fits)
       {
         gw_shadow(terrain[i].color, 76Ø, 576, 767, 583);
         stip8();
         mask(ØxcØ);
         polys(76Ø, 576, terrain[i].num_points, coords);
```

```
          mask(    f);
        }
    }
}

show_coa ()

{
  extern int current_e_coa, current_f_coa;
  static struct
   {
     long latØ, lonØ, lat1, lon1;
   } coas[6][3] =
      {
        {
          52143Ø, 1ØØØØØ, 5215ØØ, 11ØØØØ,
          5215ØØ, 1ØØØØØ, 5215ØØ, 11ØØØØ,
          52153Ø, 1ØØØØØ, 5215ØØ, 11ØØØØ
        },
        {
          521ØØØ, 1ØØØØØ, 523ØØØ, 11ØØØØ,
          5215ØØ,  95ØØØ, 5215ØØ, 111ØØØ,
          522ØØØ, 1ØØØØØ, 52ØØØØ, 11ØØØØ
        },
        {
          521ØØØ,  95ØØØ, 521ØØØ, 113ØØØ,
          521ØØØ,  95ØØØ, 521ØØØ, 113ØØØ,
          522ØØØ,  95ØØØ, 5225ØØ, 113ØØØ
        },
        {
          51593Ø, 113ØØØ, 52Ø93Ø, 1Ø3ØØØ,
          52ØØØØ, 113ØØØ, 521ØØØ, 1Ø3ØØØ,
          52ØØ3Ø, 113ØØØ, 521Ø3Ø, 1Ø3ØØØ
        },
        {
          524ØØØ, 114ØØØ, 523ØØØ, 1Ø5ØØØ,
          52ØØØØ, 1125ØØ, 521ØØØ, 11ØØØØ,
          515ØØØ, 11ØØØØ, 52ØØØØ, 1Ø3ØØØ
        },
        {
          523ØØØ, 113ØØØ, 523ØØØ, 11ØØØØ,
          52Ø5ØØ, 113ØØØ, 52Ø5ØØ, 11ØØØØ,
          514ØØØ, 112ØØØ, 52Ø5ØØ, 1Ø1ØØØ
        }
      };
   static char first_pass = 1;
   int  i, j;

   if (first_pass)
      {
        first_pass = Ø;
        for (i = Ø; i < 6; i++)
         {
           for (j = Ø; j < 3; j++)
            {
              coas[i][j].latØ = merge_deg(coas[i][j].latØ);
              coas[i][j].lonØ = merge_deg(coas[i][j].lonØ);
              coas[i][j].lat1 = merge_deg(coas[i][j].lat1);
              coas[i][j].lon1 = merge_deg(coas[i][j].lon1);
            }
          }
      }
  if (show_mask & (1 << (BLUE_UNIT+COA)))
      {
        for (i = Ø; i < 3; i++)
          {
            draw_coa(&coas[current_f_coa-1][i], BLUE);
          }
      }
  if (show_mask & (1 << (RED_UNIT+COA)))
      {
        for (i = Ø; i < 3; i++)
          {
            draw_coa(&coas[current_e_coa-1+3][i], RED);
          }
      }
}

draw_coa (coa, color)

  struct
   {
     long latØ, lonØ, lat1, lon1;
   } *coa;
  int  color;

{
  long xØ, yØ, x1, y1;

  latlon_to_pix(coa->latØ, coa->lonØ, &xØ, &yØ);
  latlon_to_pix(coa->lat1, coa->lon1, &x1, &y1);
  arrow(color, xØ, yØ, x1, y1, 6);
}

info(fmt, p1, p2, p3, p4, p5, p6, p7, p8, p9, p1Ø)

char *fmt;
long p1, p2, p3, p4, p5, p6, p7, p8, p9, p1Ø;

{
  char str[8Ø];

  sprintf(str, fmt, p1, p2, p3, p4, p5, p6, p7, p8, p9, p1Ø);
  gw_clear(infoline->w);
  gw_prints(infoline->w, str);
  cursoff();
  gw_refresh();
  curson();
}

select_1_pt(xØ, yØ, x1, y1)
```

```
int xØ, yØ, *x1, *y1;

{
  rubberband(VECT, RED, xØ  yØ, x1, y1);
}


select_2_pts(xØ, yØ, x1, y1)

int *xØ, *yØ, *x1, *y1;

{
  int oldx, oldy;

  info("Select 2 points on the map by clicking the mouse button.");

  m_hminmax(map->curx, map->curx+map->width-1);
  m_vminmax(map->cury, map->cury+map->height-1);

  while (m_posbut(xØ, yØ));

/*
  Set up a circular cursor and set the limits of the cursor to be
  the size of the available map frame.
*/

  cursoff();                              /* set up circular cursor */
  setcurs(CRCUR, 1, screen_x + *xØ, screen_y + *yØ);
  curson();

  while (!m_posbut(xØ, yØ)) {        /* move ordinary cursor until he presses */
    if (oldx != *xØ || oldy != *yØ) {
      movcurs(screen_x + *xØ, screen_y + *yØ);
      oldx = *xØ;
      oldy = *yØ;
    }
  }
  while (m_posbut(xØ, yØ)) {        /* move until he releases */
    if (oldx != *xØ || oldy != *yØ) {
      movcurs(screen_x + *xØ, screen_y + *yØ);
      oldx = *xØ;
      oldy = *yØ;
    }
  }

  cursoff();
  rubberband(VECT, RED, *xØ, *yØ, x1, y1);
  setcurs(XHCUR, 1, screen_x+*x1, screen_y+*y1);     /* Reset the cursor */
  curson();
  m_hminmax(8, screen_w-8);
  m_vminmax(8, screen_h-8);
}

rubberband(shape, color, anchorx, anchory, x, y)
```

```
int shape;
int color;
int anchorx, anchory;
int *x, *y;

{
  int oldx, oldy;

  while (m_posbut(&oldx, &oldy));
  info("Rubberband the line; click when you are done.");
  m_hminmax(map->curx, map->curx+map->width-1);
  m_vminmax(map->cury, map->cury+map->height-1);
  m_setcurs(anchorx, anchory);
  while (!m_posbut(x, y)) {
    if (*x != oldx || *y != oldy) {
      gw_refresh();
      mask(Øx3f);
      graphio(shape, color, screen_x+anchorx, screen_y+anchory, screen_x+*x,
              screen_y+*y);
      mask(Øxff);
      oldx = *x;
      oldy = *y;
    }
  }
  gw_refresh();
  m_hminmax(8, screen_w-8);
  m_vminmax(8, screen_h-8);
}


arc(color, cx, cy, radius, sa, ea)

int  color;
long cx, cy;
int  radius;
double sa, ea;

{
  double angle, step;
  long xØ, yØ, x1, y1;

  xØ = cx+cos(sa)*radius+Ø.5;
  yØ = cy+sin(sa)*radius+Ø.5;
  step = (ea-sa)/1Ø;
  for (angle = sa+step; angle < ea; angle += step)
   {
     x1 = cx+cos(angle)*radius+Ø.5;
     y1 = cy+sin(angle)*radius+Ø.5;
     clip_vectw(1, color, xØ, yØ, x1, y1);
     xØ = x1;
     yØ = y1;
   }
  x1 = cx+cos(ea)*radius+Ø.5;
  y1 = cy+sin(ea)*radius+Ø.5;
  clip_vectw(1, color, xØ, yØ, x1, y1);
```

```
}

super_high(field)
WINDOW *field;

{
     gw_fieldhue(field, field->menu->w->bgcol, WHITE);
}


super_low(field)
WINDOW *field;

{
     gw_fieldhue(field, WHITE, BLACK);
}


de_super_high()
{
 WINDOW *field;

  for (field = gw_field(friendly, 1); field; field = field->next) {
    gw_fieldhue(field, BLACK, WHITE);
  }
  for (field = gw_field(enemy, 1); field; field = field->next) {
    gw_fieldhue(field, BLACK, WHITE);
  }
}

clip_vectw(width, color, xØ, yØ, x1, y1)

  int  width, color;
  long xØ, yØ, x1, y1;

{
  double dydx, dxdy;

  if (x1 != xØ)
      {
        dydx = (double)(y1-yØ)/(x1-xØ);
      }
  if (y1 != yØ)
      {
        dxdy = (double)(x1-xØ)/(y1-yØ);
      }
  if (xØ < Ø)
      {
        if (x1 < Ø)
            {
              return;
            }
        yØ -= xØ*dydx;
        xØ = Ø;
      }
  if (xØ > 2Ø47)
```

```
      {
        if (x1 > 2Ø47)
           {
             return;
           }
        yØ -= (xØ-2Ø47)*dydx;
        xØ = 2Ø47;
      }
   if (x1 < Ø)
      {
        y1 -= x1*dydx;
        x1 = Ø;
      }
   if (x1 > 2Ø47)
      {
        y1 -= (x1-2Ø47)*dydx;
        x1 = 2Ø47;
      }
   if (yØ < Ø)
      {
        if (y1 < Ø)
           {
             return;
           }
        xØ -= yØ*dxdy;
        yØ = Ø;
      }
   if (yØ > 2Ø47)
      {
        if (y1 > 2Ø47)
           {
             return;
           }
        xØ -= (yØ-2Ø47)*dxdy;
        yØ = 2Ø47;
      }
   if (y1 < Ø)
      {
        x1 -= y1*dxdy;
        y1 = Ø;
      }
   if (y1 > 2Ø47)
      {
        x1 -= (y1-2Ø47)*dxdy;
        y1 = 2Ø47;
      }
   vectw(width, color, (int)xØ, (int)yØ, (int)x1, (int)y1);
}
```

```
#include "lvm.h"
#include "lwind.h"
#include <math.h>

#ifndef PASCAL
#define PASCAL  pascal
#endif
#define FAR     far
typedef int (FAR PASCAL *FARPROC)();

int FAR PASCAL lstrcmp(LPSTR, LPSTR);

#define abs( n ) ((n) < Ø ? -(n) : (n))
#define sqr( n ) ((n) * (n))
#define PI 3.14159265
#define EARTH_RADIUS 64ØØL

static FPOINT loc;      /* last selected map point (local coordinates) */

static int inside (lat, lon, min_lat, min_lon, max_lat, max_lon)

  long lat, lon, min_lat, min_lon, max_lat, max_lon;

{
  if ((lat < min_lat) || (lat > max_lat))
    return( Ø );
  if (min_lon <= max_lon)
    return((min_lon <= lon) && (lon <= max_lon));
  else
    return(((Ø <= lon) && (lon <= max_lon)) ||
           ((min_lon <= lon) && (lon <= 36Ø*36ØØL)));
}

pan_dir(field)

WINDOW *field;

{
  pan_map(field->param);
  pix_to_latlon((long)markx, (long)marky, &mark_lat, &mark_lon);
  reset_aoi();     /* reset the area of interest to cover the entire frame */
  redo_screen();
  return( Ø );
}


zoom_dir(field)

WINDOW *field;

{
  long x, y;

  zoom_map(field->param, mark_lat, mark_lon);
  latlon_to_pix(mark_lat, mark_lon, &x, &y);
  markx = x;
```

```
   marky = y;
   reset_aoi();
   redo_screen();
   return( Ø );
}


map_point(field, x, y, b)

   WINDOW *field;
   int  x, y, b;

{
   markx = x;
   marky = y;
   pix_to_latlon((long)markx, (long)marky, &mark_lat, &mark_lon);
   cursoff();
   gw_movwin(mapmark, markx-mapmark->width/2, marky-mapmark->height/2);
   gw_refresh();
   curson();
   return( Ø );
}

toggle_units (field)

   WINDOW *field;

{
   show_mask ^= field->param;
   redo_screen();
   return( Ø );
}

clear_screen (field)

   WINDOW *field;

{
   show_mask = Ø;
   redo_screen();
   return( Ø );
}

return_pc()

{
   return(1);
}


redo_screen ()

{
   if (inpw(Øx3eØ) == -1)
      {
        return;
```

```
     }
   gw_movwin(mapmark, markx-mapmark->width/2, marky-mapmark->height/2);
   gw_getvid();
   clip_aoi();
   show_icons();
   show_coa();
   unclip_aoi();
   show_terrain();
   gw_repaint();
   draw_aoi();
   cursoff();
   gw_refresh();
   curson();
   return( Ø );
 }


show_icons ()

 {
   int draw_icon();  /* A routine which is called when 'do_icon_region'
                        finds an icon that falls within a region of
                        interest (in lvmdraw.c) */

   extern long split_deg();

   if (!show_mask) return;     /* no icons currently shown .... */

 /* Display the icons that fall within the area of interest */

   do_icon_region(aoi_lat1, aoi_lon1, aoi_lat2, aoi_lon2, draw_icon);
   return( Ø );
 }

 area_of_interest(field, x, y, b)

   WINDOW *field;
   int  x, y, b;

 {
   menu_toggle();
   get_aoi(x, y);         /* in lvmdraw.c */
   menu_toggle();
   redo_screen();
   return( Ø );
 }

 select_aoi()

 {
   int x, y;
   hijack_mouse();
   m_posbut(&x, &y);
   area_of_interest(Ø, x, y);
   cursoff();
```

```
  return_mouse();


menu_toggle()

  static char func_on = 1;

  cursoff();
  gw_clear(plan->w);
  if (func_on = !func_on) {
    gw_popup(functions, 15, 37);
    gw_prints(plan->w," Menu Off");  /* Function now allows em to turn it off */
  }
  else {
    gw_remove(functions);
    gw_prints(plan->w," Menu On");
  }
  gw_refresh();
  curson();
  return( Ø );
}



turn_off_self(field)

INDOW *field;

{
  gw_remove(field->menu);
  gw_refresh();
  return(Ø);




eset_aoi ()
{
  aoi_x1 = Ø;
  aoi_y1 = Ø;
  aoi_x2 = screen_w - 1;
  aoi_y2 = screen_h - 1;
  pix_to_latlon(ØL, ØL, &aoi_lat1, &aoi_lon1);
  pix_to_latlon((long)(screen_w-1), (long)(screen_h-1), &aoi_lat2, &aoi_lon2);
}



how_distance()

{
  int xØ, yØ, x1, y1;
  long latØ, lonØ, lat1, lon1;
  double a, b, c, d2, e, f;     /* sorry, but I don't have names for them */
  double angle, distance;

  info("Select 2 points on the map by clicking the mouse button.");
  menu_toggle();
  select_2_pts(&xØ, &yØ, &x1, &y1);
  menu_toggle();
  pix_to_latlon((long)xØ, (long)yØ, &latØ, &lonØ);
  pix_to_latlon((long)x1, (long)y1, &lat1, &lon1);
  a = 2*sin((double)labs(lon1-lonØ)/36ØØ*PI/18Ø.Ø/2);
  b = tan((double)latØ/36ØØ*PI/18Ø.Ø);
  c = tan((double)lat1/36ØØ*PI/18Ø.Ø)-b;
  d2 = sqr(a)+sqr(c);   /* d would be sqrt(d2), but only d2 is needed */
  e = 1/cos((double)latØ/36ØØ*PI/18Ø.Ø);
  f = 1/cos((double)lat1/36ØØ*PI/18Ø.Ø);
  angle = acos((sqr(e)+sqr(f)-d2)/(2*e*f));
  distance = EARTH_RADIUS*angle;
  info("The distance is %.2lf kilometers", distance);
  return( Ø );
}

describe_icon (index)

  long index;

{
  static char description[8Ø];

  if (icons[index].class >= RED_UNIT)
    {
      strcpy(description, "ENEMY ");
    }
   else
    {
      strcpy(description, "FRIENDLY ");
    }
  switch (icons[index].class & 7)
   {
     case BRIGADE:
          strcat(description, "BRIGADE");
          break;
     case CORPS:
          strcat(description, "CORPS");
          break;
     case HQ:
          strcat(description, "HEAD QUARTERS");
          break;
     case REGIMENT:
          strcat(description, "REGIMENT");
          break;
     case DIVISION:
          strcat(description, "DIVISION");
          break;
   }
  info(description);
}
```

```
get_data()

{
  long latØ, lonØ, lat1, lon1;
  long split_deg();

  pix_to_latlon((long)(markx-1Ø), (long)(marky-1Ø), &latØ, &lonØ);
  pix_to_latlon((long)(markx+1Ø), (long)(marky+1Ø), &lat1, &lon1);
  do_icon_region(latØ, lonØ, lat1, lon1, describe_icon);
  return(Ø);
}


static char plan_on = Ø;

plan_el_select(field)

WINDOW *field;

{
  gw_remove(planel);
  plan_on = Ø;
  gw_popup(menu[field->param], plan->curx+gw_field(plan,1)->bx,
           screen_h - 16 - menu[field->param]->height);
  gw_refresh();
  return(Ø);
}


plan_toggle()

{
  cursoff();
  if (plan_on = !plan_on) {
    gw_popup(planel, plan->curx+gw_field(plan, 1)->bx,
             plan->cury-planel->height);
  }
  else {
    gw_remove(planel);
  }
  gw_refresh();
  curson();
  return( Ø );
}
```

```
#include "parallax.h"

#define extern
#include "lvm.h"
#undef extern

lvminit(map_directory)

  char *map_directory;

{
  int oldind = Ø;      /* old cursor table index (initially points to crosshair */
  int oldx, oldy;      /* old mouse cursor position values */
  char digitize, key;
  char small;
  int i;

  static char *clist[] = {"xhcur.p", "swcur.p", "scur.p", "secur.p", "wcur.p",
                          "ecur.p", "nwcur.p", "ncur.p", "necur.p", "circle.p"};

  if (inpw(Øx3eØ) == -1)
     {
       return;
     }
  small = Ø;
  digitize = Ø;
  graphini(1);      /* initialize graphics */
  pan(screen_x,screen_y+479);
  zoom(2, 2);
  gw_loadall("menus.lst");
  gw_varini();                 /* Load the menus into memory */
  ini_curstab(1Ø);             /* Load the cursor fonts into memory */
  for (i = Ø; i < 1Ø; i++)
   {
     defcur(i, RED, clist[i]);
   }
  gw_l_icon_imgtab("icons.lst", &icontab); /* Load the icon fonts into memory */

  gw_popup(asof, 44Ø, 4ØØ);
  gw_popup(infoline, 16, 16);
  gw_popup(rim, Ø, Ø);
  gw_popup(map, 16, 16);
  for (i = Ø; i < 4; i++) {
    gw_popup(in[i], inx[i], iny[i]);
    gw_popup(out[i], outx[i], outy[i]);
  }
  gw_popup(functions, 16, 37);
  gw_popup(plan, 15, 444);
  gw_prints(plan->w, " Menu Off");

 /* Initialize the video mapping module data base */

  load_vidmap(map_directory);

 /* Load up the icon data base (from VAX) into memory */
```

```
    load_icons("icons.db");

    reset_aoi();            /* area of interest for displaying icons */

    markx = screen_w/2;
    marky = screen_h/2;
    m_hminmax(8,screen_w-8);
    m_vminmax(8,screen_h-8);
    gw_popup(mapmark, markx-mapmark->width/2, marky-mapmark->height/2);
    show_mask = Ø;
    redo_screen();
}
```

```
#include        <stdio.h>
#include        <dos.h>

static union REGS regs86;

        wait(ticks)
        /*====================================================*/
        /*      Wait for the specified number of ticks.        */
        /*----------------------------------------------------*/
        /*    Time is expressed as a number of clock ticks.    */
        /*    (On the IBM PC there are 18.2 ticks/second.)     */
        /*====================================================*/
        unsigned        ticks;
        {
            long        time, etime;

            regs86.h.ah = Ø;            /* Get time service code */
            int86(Øx1A,&regs86,&regs86);
            time = (long)((regs86.x.cx << 16) + regs86.x.dx);
            etime = time + ticks;
            while (time <= etime)
              {
                regs86.h.ah = Ø;
                int86(Øx1A,&regs86,&regs86);
                time = (long)((regs86.x.cx << 16) + regs86.x.dx);
              }
        }
```

```
#include "parallax.h"
#include "boundary.h"
#include "lvm.h"        /* includes gwindows.h */

#define extern
#include "gwvars.h"
#undef extern

int screen_x = Ø;
int screen_y = 512;
int screen_w = 64Ø;
int screen_h = 48Ø;

int chsizh = 16;        /* horizontal size of a character in pixels */
int chsizv = 32;        /* vertical size of a character in pixels */

int zoomx = 2;
int zoomy = 2;

gw_varini()

{
  int i;
  int zoom_dir(), map_point(), pan_dir(), toggle_units();
  int return_pc(), menu_toggle(), area_of_interest();
  int show_distance(), clear_screen(), get_data();
  int turn_off_self();

  WINDOW *field;

  rim = menu[Ø];
  for (field = rim->w; field; field = field->next) {
    field->function = pan_dir;
  }
  map = menu[1];
  map->w->function = map_point;
  for (i = Ø; i < 4; i++) {
    in[i] = menu[i+2];
    in[i]->w->function = zoom_dir;
    out[i] = menu[i+6];
    out[i]->w->function = zoom_dir;
  }
  inx[Ø] = inx[2] = INLEFT1;
  iny[Ø] = iny[1] = Ø;
  inx[1] = inx[3] = INRIGHT1;
  iny[2] = iny[3] = MAXY - BORDER;
  outx[Ø] = outx[2] = Ø;
  outy[Ø] = outy[1] = OUTBOT1;
  outx[1] = outx[3] = MAXX - BORDER;
  outy[2] = outy[3] = OUTTOP1;
  asof = menu[1Ø];
  functions = menu[11];
  gw_field(functions,Ø)->function = return_pc;
  gw_field(functions,1)->function = get_data;
  gw_field(functions,2)->function = show_distance;
  gw_field(functions,3)->function = clear_screen;
```

```
    gw_field(functions,4)->function = area_of_interest;
    gw_field(functions,5)->function = toggle_units;
    gw_field(functions,6)->function = toggle_units;
    infoline = menu[12];
    plan = menu[13];
    gw_field(plan,Ø)->function = menu_toggle;
    mapmark = menu[14];
    mapmark->w->function = map_point;
}
```

```
#include "lvm.h"        /* defines the icon structure and pointer to     */

#define min(a,b) (a < b ? a : b)

typedef struct
 {
   long lat;              /* existing LAT value from icon database */
   long index;            /* index of first icon in database with this X value */
 } x_index_struct;        /* states starting index for particular X value */

long num_icons;           /* number of icons currently in the database */
long max_icons;           /* maximum number of icons the database can hold */
unsigned num_x_values;    /* number of distinct X values in the database */
unsigned max_x_values;    /* maximum number of x_index entries we can hold */
x_index_struct *x_indexes;/* beginnings of distinct X values in the database */
                            /* allocated at run-time */
/* char *lsbrk(); */
char *malloc();

movlmem (source,destination,amount)

  char *source, *destination;
  long amount;

{
  if (source < destination)
     {
       source += amount;
       destination += amount;
       while (amount > 30000)
        {
          amount -= 30000;
          source -= 30000;
          destination -= 30000;
          memcpy(destination,source,30000);
        }
       source -= amount;
       destination -= amount;
       memcpy(destination,source,(int)amount);
     }
    else
     {
       while (amount > 30000)
        {
          memcpy(destination,source,30000);
          amount -= 30000;
          source += 30000;
          destination += 30000;
        }
       memcpy(destination,source,(int)amount);
     }
 }

load_icons (filename)
```

```
  char *filename;

{
  long database_size, index, last_x;
  int  bytes_read;
  char *ptr;
  int  handle;

  handle = open(filename,Øx8ØØØ);
  /* read num_icons, if error assume no previous data */
  if (read(handle,&num_icons,sizeof num_icons) != sizeof num_icons)
     {
       num_icons = Ø;
     }
  max_icons = num_icons+2ØØ;  /* allow for 2ØØ insertions */
  database_size = max_icons*(sizeof *icons);    /* space to alloc. for d.b. */
  icons = (icon_struct *)malloc((int)database_size);  /* request memory from DOS
 */
  if (num_icons)
     {
       database_size = num_icons*(sizeof *icons);   /* amount of prev. data */
       ptr = (char *)icons;
        do {                                   /* read in bursts of 4k */
          bytes_read = read(handle,ptr,(int)min(database_size,4Ø96));
          ptr += bytes_read;
          database_size -= bytes_read;
        } while (database_size && bytes_read);  /* until EOF or end of d.b. */
     }
  close(handle);
  num_x_values = Ø;               /* count num_x_values */
  last_x = icons[Ø].lat-1;       /* make sure last_x := icons[Ø].lat */
  for (index = Ø; index < num_icons; index++)   /* scan whole d.b. */
   {
     if (last_x != icons[index].lat)      /* new X? */
        {
          last_x = icons[index].lat;      /* remember this value */
          num_x_values++;                /* count this value */
        }
   }
  max_x_values = num_x_values+1ØØ;      /* allow for 1ØØ new X values */
  database_size = (max_x_values+1)*(sizeof *x_indexes); /* size of x_indexes */
  x_indexes = (x_index_struct *)malloc((int)database_size); /* get memory from D
OS */
  num_x_values = Ø;                        /* build x_indexes */
  last_x = icons[Ø].lat-1;                     /* make sure last_x != icons[Ø].lat */
  for (index = Ø; index < num_icons; index++)   /* scan whole d.b. */
   {
     if (last_x != icons[index].lat)      /* new X? */
        {
          last_x = icons[index].lat;      /* remember this value */
          x_indexes[num_x_values].lat = last_x; /* record this X value & index *
/
          x_indexes[num_x_values++].index = index;  /* and increment counter */
        }
   }
  x_indexes[num_x_values].lat = Øx7FFFFFFF;   /* last entry+1 points to end */
  x_indexes[num_x_values].index = num_icons;
}

store_icons (handle)

  int  handle;

{
  long database_size;
  int  bytes_written;
  char *ptr;

  write(handle,&num_icons,sizeof num_icons);     /* write num_icons */
  database_size = num_icons*(sizeof *icons);     /* amount of data to write */
  ptr = (char *)icons;
  while (database_size) /* write in bursts of 4k, since write limited to 64k */
   {
     bytes_written = write(handle,ptr,(int)min(database_size,4Ø96));
     database_size -= bytes_written;
     ptr += bytes_written;
   }
}

unsigned find_x_index (x)

  long x;

{
  unsigned low_end, high_end, middle;

  /* perform binary search on x_indexes, and return the resulting index */
  low_end = Ø;
  high_end = num_x_values;
  while (((middle = (low_end+high_end)/2) != low_end) &&
         (x_indexes[middle].lat != x))
   {
     if (x_indexes[middle].lat < x)
        {
          low_end = middle;
        }
       else
        {
          high_end = middle;
        }
   }
  if (x_indexes[middle].lat < x)
     {
       middle++;
     }
  return(middle);
}

long find_icon (x_index,y)

  unsigned x_index;
  long y;
```

```
{
  long low_end, high_end, middle;

  /* perform binary search on range of icons specified by x_indexes[x_index],
     and x_indexes[x_index+1], and return the resulting index */
  low_end = x_indexes[x_index].index;
  high_end = x_indexes[x_index+1].index;
  while (((middle = (low_end+high_end)/2) != low_end) &&
         (icons[middle].lon != y))
   {
     if (icons[middle].lon < y)
       {
         low_end = middle;
       }
      else
       {
         high_end = middle;
       }
   }
  if (icons[middle].lon < y)
     {
       middle++;
     }
  return(middle);
}

do_icon_region (min_lat, min_lon, max_lat, max_lon, handler)

  long min_lat, min_lon, max_lat, max_lon;
  int (*handler)();

{
  unsigned lat_index;
  long icon_index;

  if (min_lon > max_lon) min_lon -= 360*3600L;
  lat_index = find_x_index(min_lat);         /* find first desired range */
  while (x_indexes[lat_index].lat < max_lat)  /* scan all potential ranges */
   {
     icon_index = find_icon(lat_index,min_lon);      /* find first desired icon *
/
     while ((icons[icon_index].lon < max_lon) &&     /* scan all desired icons */
            (icon_index < x_indexes[lat_index+1].index))
      {
        (*handler)(icon_index);       /* call the handler, passing icon_index */
        icon_index++;                 /* proceed to the next icon */
      }
     if (min_lon < 0) {
       min_lon += 360*3600L;
       icon_index = find_icon(lat_index,min_lon);
       while ((icons[icon_index].lon < 360*3600L) &&
              (icon_index < x_indexes[lat_index+1].index))
        {
          (*handler)(icon_index);       /* call the handler, passing icon_index *
/
```

```
           icon_index++;                   /* proceed to the next icon */
         }
       min_lon -= 36Ø*36ØØL;
      }
     lat_index++;                          /* proceed to the next X value */
    }
}

insert_icon (x, y, class)

   long x, y;
   unsigned class;      /* classification of type of icon */

{
   unsigned x_index;
   long icon_index, shift_size;
   char *ptr;

   if (num_icons >= max_icons)   /* room for new icon? */
      {
        return;
      }
   x_index = find_x_index(x);    /* position of new icon in x_indexes */
   if (x_indexes[x_index].lat != x)        /* new X value? */
      {
        if (num_x_values >= max_x_values)        /* room for new X value? */
           {
             return;
           }
        /* shift front end of x_indexes forward to make room for new X value */
        shift_size = (num_x_values+1-x_index)*((long)sizeof *x_indexes);
        movlmem(&x_indexes[x_index],&x_indexes[x_index+1],shift_size);
        num_x_values++;           /* update num_x_values */
        x_indexes[x_index].lat = x;/* load new X value into x_indexes, index OK *
/
        icon_index = x_indexes[x_index].index;   /* place to insert in icons */
      }
     else
      {
        icon_index = find_icon(x_index,y);       /* place to insert in icons */
      }
   /* shift front end of icons forward to make room for new icon */
   shift_size = (num_icons-icon_index)*(sizeof *icons);
   movlmem(&icons[icon_index],&icons[icon_index+1],shift_size);
   num_icons++;           /* update num_icons */
   icons[icon_index].lat = x;      /* load new icon into his home */
   icons[icon_index].lon = y;
   icons[icon_index].class = class;
   /* update x_indexes to reflect the shift in icons */
   while (++x_index <= num_x_values)
    {
      x_indexes[x_index].index++;
    }
}

delete_icon (x,y)
```

```
  long x, y;

{
  unsigned x_index, index;
  long icon_index, shift_size;

  x_index = find_x_index(x);              /* find range of X of specified icon */
  if (x_indexes[x_index].lat != x)          /* no such X -> no such icon */
     {
       return;
     }
  icon_index = find_icon(x_index,y);     /* find specified icon */
  if ((icons[icon_index].lon != y) || (icon_index >= x_indexes[x_index+1].index)
)
     {
       return;                            /* icon doesn't exist */
     }
  shift_size = (num_icons-icon_index-1)*(sizeof *icons);      /* delete icon */
  movlmem(&icons[icon_index+1],&icons[icon_index],shift_size);
  num_icons--;                            /* one less icon in list */
  index = x_index;      /* update x_indexes to reflect shift in icons */
  while (++index <= num_x_values)
   {
     x_indexes[x_index].index--;
   }
  /* if the deleted icon was the last with that X value, zap x_indexes entry */
  if (x_indexes[x_index].index == x_indexes[x_index+1].index)
     {
       /* delete the entry by shifting the front end backwards */
       shift_size = (num_x_values-x_index)*(long)(sizeof *x_indexes);
       movlmem(&x_indexes[x_index+1],&x_indexes[x_index],shift_size);
       num_x_values--;           /* one less existing X value */
     }
}
```